Package 'TempleCBE'

October 14, 2026

Type Package

Title Temple CBE Biostatistics, Clinical Data Science, and Modeling Utilities

Version 0.5.2

Description An open-source R package developed for Temple University's Center for Biostatistics and Epidemiology (CBE). Provides domain-agnostic biostatistical testing, data quality and missingness visualizations, normalization functions, custom Tidymodels recipe steps, and clinical survival metrics.

License GPL-3 | MIT + file LICENSE

Encoding UTF-8

URL https://github.com/jkylearmstrong/TempleCBE, https://jkylearmstrong.github.io/TempleCBE/

BugReports https://github.com/jkylearmstrong/TempleCBE/issues

Depends R (>= 4.2.0)

Imports broom,
corrplot,
dplyr (>= 1.1.0),
exact2x2,
fs,
generics,
ggplot2 (>= 3.5.0),
ggridges (>= 0.5.0),
ggsurvfit,
glue,
gtsummary,
hardhat (>= 1.3.0),
here,
grDevices,
grid,
igraph,
janitor,
labelled,
methods,
nortest,
openssl,
pdftools (>= 2.0),
purrr (>= 1.0.0),
r2rtf,

readxl,
recipes (>= 1.0.0),
rlang (>= 1.1.0),
scales (>= 0.5.0),
stats,
stringr,
survival (>= 3.5.0),
tibble (>= 3.2.0),
tidygraph,
tidyr (>= 1.3.0),
tools,
utils,
vctrs (>= 0.6.0),
writexl,
xml2

Suggests arsenal,
baguette (>= 1.1.0),
BradleyTerry2,
callr,
caret,
censored (>= 0.3.0),
DALEX (>= 2.5.0),
DALEXtra (>= 2.3.0),
devtools,
dials,
DiagrammeR,
embed (>= 1.1.0),
FactoMineR,
finetune (>= 1.1.0),
furrr (>= 0.2.0),
future,
ggpubr,
ggraph,
ggVennDiagram,
glmnet (>= 4.1.0),
gt,
haven,
htmlwidgets,
jsonlite,
knitr,
missForest,
missRanger (>= 2.4.0),
openxlsx (>= 4.2.6),
parsnip,
pkgload,
probably (>= 1.2.0),
profvis,
quarto (>= 1.4),
reticulate,
rmarkdown,
rsample (>= 1.1.0),
stacks (>= 1.1.1),

survex (>= 1.2.0),
testthat (>= 3.2.0),
tidymodels (>= 1.1.0),
tune (>= 1.1.0),
visNetwork,
withr (>= 2.3.0),
workflows,
workflowsets (>= 1.1.0),
yaml,
yardstick (>= 1.3.0),
zip (>= 2.3.0)

VignetteBuilder knitr

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

Contents

add_graf_weights . . . 8
add_missing . . . 9
anonymize_pi . . . 10
apply_database_metadata . . . 12
as_database . . . 13
as_igraph.cbe_database_relationships . . . 14
as_pipeline_graph . . . 14
as_tbl_graph.list . . . 15
audit_report_deliverables . . . 15
autoplot.cbe_compare_df . . . 16
autoplot.cbe_database_relationships . . . 16
autoplot.cbe_key_integrity . . . 17
autoplot.cbe_loco_mp_coxnet . . . 17
autoplot.cv_joint_model . . . 18
build_manual_versioned . . . 18
calculate_fences . . . 19
cbe_balloon_plot . . . 19
cbe_bar_plot . . . 20
cbe_compare_df . . . 21
cbe_contingency_plot . . . 24
cbe_cox_check . . . 28
cbe_cox_multi . . . 29
cbe_cox_single . . . 30
cbe_cox_table . . . 31
cbe_database_relationships . . . 32
cbe_database_venn . . . 33
cbe_docx_review_extract . . . 34
cbe_exact2x2 . . . 37
cbe_exact2x2_ci . . . 39
cbe_explain . . . 40
cbe_explain_survival . . . 41
cbe_factor_reference . . . 42
cbe_four_quadrant_report . . . 43
cbe_heatmap_plot . . . 44

cbe_km_single . . . 45
cbe_labels . . . 46
cbe_loco_mp_coxnet . . . 47
cbe_mosaic_plot . . . 50
cbe_pairwise_combos . . . 51
cbe_palette . . . 52
cbe_predict_parts_shap . . . 52
cbe_sas_macro_dir . . . 53
cbe_sas_macro_path . . . 54
cbe_square_plot . . . 54
cbe_survex_loss_ibs . . . 56
cbe_test_categorical . . . 57
cbe_theme_survival . . . 59
cbe_variable_roles . . . 59
censoring_km . . . 61
check_docx_toolchain . . . 62
clean_names . . . 63
clean_publish . . . 63
collapse_by_stage . . . 68
compute_graph . . . 69
controls_datasets . . . 69
control_race_survival . . . 70
convert_pdfs_to_docx . . . 71
convert_pdf_to_docx . . . 72
copy_data_manifest . . . 73
correlation_diff . . . 74
correlation_diff_heatmap . . . 75
correlation_plot . . . 76
correlation_plot_split . . . 77
corr_test_all . . . 78
coxnet . . . 79
coxnet_train . . . 81
create_qmd_renderer . . . 83
create_report . . . 83
create_toc_from_sas_pdf . . . 85
cv_coxnet . . . 86
cv_coxnet-methods . . . 90
cv_joint_model . . . 91
database_metadata_io . . . 92
database_setops . . . 93
database_to_igraph . . . 95
delete_nul_files . . . 96
demo_cbe_mapping . . . 96
detect_outliers . . . 97
distribution_plot . . . 97
distribution_test . . . 98
export_interactive_pipeline . . . 98
export_subgraph . . . 99
extract_all_xlsx_tokens . . . 100
extract_surv_components . . . 100
extract_win_posix_paths . . . 101
extract_yaml_formats . . . 102

features_percent_miss . . . 102
FileOutputs . . . 103
FileOutputs-class . . . 103
FilePath . . . 104
FilePath-class . . . 105
FileUses . . . 106
FileUses-class . . . 107
file_meta_fs . . . 107
find_code . . . 108
find_correlation . . . 109
find_python . . . 110
find_sas . . . 111
find_section_file . . . 112
find_soffice . . . 112
fit_n_rank . . . 113
flag_outliers . . . 114
fmt_hr . . . 114
fmt_num . . . 115
fmt_p . . . 115
fmt_pct . . . 116
fmt_sig . . . 116
generate_last_names . . . 117
generate_pi_names . . . 117
generate_pseudonym_token . . . 118
get_dataset_info . . . 120
get_model_parameters . . . 121
get_render_plan . . . 122
glmnet_IBS . . . 123
graf_weights . . . 125
graph_setops . . . 126
guard_public_remote . . . 127
infix_helpers . . . 128
install.packages.no_lock . . . 129
is.int . . . 130
is_normal . . . 130
is_poisson . . . 131
joint_model . . . 132
join_pipelines . . . 134
keep_only . . . 135
km_summary_to_prism . . . 135
list_stage_data_artifacts . . . 137
make_excel_names . . . 137
manhattan_plot . . . 138
min_max_norm . . . 138
missforest_impute_by_mtry . . . 139
missforest_oob_by_mtry . . . 140
missforest_sweep_mtry . . . 141
missmap . . . 143
missranger_max_mtry . . . 144
missranger_oob_by_mtry . . . 145
missranger_sweep_mtry . . . 147
multiple_t_test . . . 150

my_summary_table . . . 151
nested_cv_coxnet . . . 151
nested_cv_joint_model . . . 154
normalize_safely . . . 155
one_vs_rest_t_test . . . 156
package_deliverables . . . 157
parse_here_call_vec . . . 158
pca_biplot . . . 158
pca_eqns . . . 159
pca_feature_loading_heatmap . . . 160
pca_loading_diff . . . 160
pca_loading_diff_heatmap . . . 161
pca_percent_var_explained . . . 162
pca_plot . . . 162
pca_scree_plot . . . 163
pca_variables_plot . . . 164
pdf_to_rtf . . . 165
pformat . . . 166
pipeline_config . . . 167
pipeline_summary . . . 167
pi_anonymizer . . . 168
plot.features_percent_miss . . . 169
plot_categorical_associations . . . 169
plot_cox_forest . . . 171
plot_cox_forest_multi . . . 172
plot_cox_marginal . . . 173
plot_cox_survival . . . 174
plot_dynamic_trajectory . . . 175
plot_group_comparison . . . 175
plot_missingness . . . 176
plot_pca_bi . . . 177
plot_survival_km . . . 177
predict.coxnet_model . . . 178
predict.joint_model . . . 179
print.cbe_cox . . . 180
print.cbe_cox_check . . . 180
print.cbe_cox_multi . . . 181
print.cbe_km . . . 181
print.joint_model . . . 181
print_pipeline . . . 182
proc_pca . . . 183
profvis_summary . . . 183
racing_workflows . . . 184
range_norm . . . 185
read_database_from_excel . . . 185
read_data_manifest . . . 186
read_excel_multiple_headers . . . 187
read_mapped_section_data . . . 188
read_raw_table . . . 189
read_search . . . 189
read_workbook . . . 190
render . . . 190

review_config . . . 192
rossi_data . . . 194
rotation_matrix . . . 195
run_sas_script . . . 196
R_names . . . 197
scale_color_cbe . . . 197
scale_colour_temple . . . 198
scale_fill_cbe . . . 199
scan_data_io . . . 199
sd.error . . . 200
significance_stars . . . 201
simulate_section_data . . . 201
single_t_test . . . 202
step_famd . . . 203
step_lencode_coxnet . . . 205
step_lencode_joint_model . . . 207
stop_if_invalid_manifest . . . 209
summarize_section_by_time . . . 209
summarize_tune_results . . . 210
summary.tbl_graph . . . 211
SumNa . . . 211
surv_subject_truth . . . 212
table_two_by_two . . . 213
temple_brand_path . . . 214
temple_colors . . . 214
temple_pal . . . 215
theme_cbe . . . 215
theme_cbe_deck . . . 216
theme_temple . . . 216
tidy.cbe_loco_mp_coxnet . . . 217
tidy.coxnet_model . . . 217
tidy.joint_model . . . 218
tidy_tmerge_cox . . . 218
tumor_long . . . 220
tumor_wide . . . 221
tune_over_alpha . . . 222
tune_race_survival . . . 223
use_temple_brand . . . 225
validate_column_mapping . . . 226
validate_data_manifest . . . 226
visualize_pipeline . . . 227
visualize_pipeline_interactive . . . 228
volcano_plot . . . 228
words . . . 229
write_database_to_excel . . . 230
write_search . . . 231
zip_render . . . 232
zip_reports . . . 233
z_norm . . . 234

Index **236**

`add_graf_weights` *Add Censoring Weights to Survival Predictions for yardstick*

Description

Fills the ‘.weight_censored‘ column of each element of a ‘.pred‘ list-column with [graf_weights()], so the data can go straight to ‘yardstick::brier_survival()‘, ‘brier_survival_integrated()‘, or ‘roc_auc_survival()‘. Use it to score any model's per-subject survival predictions, including models fit to start/stop data.

Usage

```
add_graf_weights(
  data,
  truth = ".truth",
  estimate = ".pred",
  censoring,
  trunc = 0.05
)
```

Arguments

`data`	A data frame with one row per subject.
`truth`	Column of ‘data‘ holding right-censored ‘Surv‘ truth.
`estimate`	List-column of data frames with ‘.eval_time‘ and ‘.pred_survival‘, one data frame per row, all with the same evaluation times.
`censoring`	A [censoring_km()] object, estimated on the training data.
`trunc`	Lower bound for G, capping weights at ‘1 / trunc‘ (default 0.05, as in tidymodels' ‘parsnip‘).

Value

‘data‘, with ‘.weight_censored‘ added to (or replaced in) every data frame in ‘estimate‘.

See Also

[surv_subject_truth()], [censoring_km()]

Examples

```
if (requireNamespace("survival", quietly = TRUE) &&
    requireNamespace("yardstick", quietly = TRUE)) {
  train <- survival::Surv(c(2, 4, 6, 8), c(0, 1, 0, 1))
  scored <- tibble::tibble(
    .truth = survival::Surv(c(5, 7), c(1, 0)),
    .pred = list(
      tibble::tibble(.eval_time = c(3, 6), .pred_survival = c(0.9, 0.5)),
      tibble::tibble(.eval_time = c(3, 6), .pred_survival = c(0.8, 0.6))
    )
  )
  scored <- add_graf_weights(scored, censoring = censoring_km(train))
  yardstick::brier_survival(scored, truth = .truth, .pred)
}
```

`add_missing` *Add Missing Values Completely at Random*

Description

Sets a fixed share of cells in each selected column to `NA`, choosing the rows independently for each column (missing completely at random). Use it to test imputation: start from complete data, add missingness, impute, and compare the imputed cells with the originals.

Usage

```
add_missing(data, cols = dplyr::everything(), pct_na = 0.1)
```

Arguments

`data`	A data frame or tibble.
`cols`	Columns to add missing values to, as a tidyselect expression (default `dplyr::everything()`).
`pct_na`	Proportion of cells to set to `NA` in each selected column, between 0 and 1 (default 0.1). Can be a single number applied to all selected columns, or a numeric vector with length equal to the number of selected columns specifying column-specific amputation rates.

Details

Every selected column must start with no missing values, so afterwards its share of `NA`s equals `pct_na` (to the nearest whole row), matching `PctNa` from `features_percent_miss`. The observed share is `1 - pct_na`, so only `pct_na` is given.

Value

`data` with missing values added. Its `"missing_cells"` attribute is a tibble with one row per masked cell, giving its `row` number and `feature` (column name). Most dplyr verbs drop this attribute, so read it before transforming the result.

Examples

```
set.seed(1)
amputed <- add_missing(mtcars, c(mpg, hp), pct_na = 0.25)
features_percent_miss(amputed)
head(attr(amputed, "missing_cells"))

# Column-specific missingness proportions with a vector:
p_vec <- c(mpg = 0.1, hp = 0.3)
amputed2 <- add_missing(mtcars, c(mpg, hp), pct_na = p_vec)
features_percent_miss(amputed2)
```

anonymize_pi *Pseudonymize Investigator Names (Vectorized)*

Description

Replaces investigator names with pseudonyms. Supports both single scalar names and character vectors (e.g. data frame columns like `df$investigator`). Offers stateless keyed-HMAC hashing as well as a stateful mapping table.

Usage

```
anonymize_pi(
  name,
  method = c("hmac_token", "hmac_surname", "token", "synthetic"),
  key = Sys.getenv("TEMPLECBE_SECRET_KEY", unset = ""),
  secrets_path = NULL,
  exclude = character(),
  prefix = "PI_",
  n_chars = 16,
  rename_fn = NULL,
  seed = NULL,
  allow_default_key = FALSE,
  allow_collisions = FALSE,
  normalize = FALSE
)
```

Arguments

`name`	Character vector of investigator names or surnames to mask.
`method`	Character string specifying the approach: `"hmac_token"`, `"hmac_surname"`, `"token"`, or `"synthetic"`.
`key`	Character string, the secret key for the HMAC methods (and for keyed tokens with `method = "token"`). Defaults to the environment variable `TEMPLECBE_SECRET_KEY`. An empty key counts as no key. Prefer the environment variable over writing the key into a script.
`secrets_path`	Path to the confidential mapping file when `method = "token"`. Defaults to `tools::R_user_dir("Tem` `which = "data")/pi_mapping.json`. It must be outside any Git repository.
`exclude`	Character vector of names that must never be generated (passed to `generate_last_names` when method is `"hmac_surname"` or `"synthetic"`).
`prefix`	Character string, prefix for tokens (default `"PI_"`).
`n_chars`	Integer, number of hex characters for tokens (default 16). Can be shortened for compact tokens; very short tokens (e.g. 4) collide, so different investigators may share one.
`rename_fn`	Optional function to rename or format generated tokens/pseudonyms.
`seed`	Optional integer, random seed when method is `"synthetic"`.
`allow_default_key`	Logical, default `FALSE`. If `TRUE` and no key is available for `"hmac_token"` or `"hmac_surname"`, use the package's public built-in key instead of stopping, and warn that the output is NOT secret. This only reproduces output made without a key by earlier versions; do not use it for data you intend to protect.

`allow_collisions`
: Logical, default `FALSE`. If `TRUE`, suppresses collision errors/warnings when distinct names produce identical tokens or surnames.

`normalize`
: Logical, default `FALSE`. If `TRUE`, trims leading/trailing whitespace and converts names to lowercase before hashing.

Details

Scope. Only investigator names are replaced. Dates, record or subject numbers, free text, and other identifiers are left untouched, so this is not a general de-identification tool; see `pi_anonymizer` for the limits.

Key required for the deterministic methods. `"hmac_token"` and `"hmac_surname"` stop with an error unless a secret key is available, from the `key` argument or the `TEMPLECBE_SECRET_KEY` environment variable. Without a secret key anyone can recompute the pseudonyms from a list of names. Set the variable in your user '`.Renviron`' rather than writing the key into a script that may be committed. Passing `allow_default_key = TRUE` restores the earlier fallback to a public built-in key, with a warning that the output is NOT secret. The check happens before the data is looked at, so an all-`NA` input does not skip it.

Methods:

- `"hmac_token"` (default): Stateless deterministic token (e.g., `PI_5c1ebfd9a0b3c4d2`) computed via `openssl::sha256(name, key = key)`. Needs a key. No mapping file is written.
- `"hmac_surname"`: Stateless deterministic procedural surname generated via a keyed HMAC seed and `generate_last_names`. Needs a key. Consistent across runs without disk state; different names can share a surname.
- `"token"`: Stateful tokens persisted to a user-level JSON mapping file (`tools::R_user_dir("TempleCBE", "data")/pi_mapping.json`), so a name keeps its token between calls. Works without a key: new names then get random tokens, which exist only in the mapping file. If a key is set, tokens are the keyed HMAC of the name instead. The mapping file lists real names next to their tokens and is confidential. The function refuses to create or write it inside a Git repository tree, to prevent accidental commits.
- `"synthetic"`: Random procedural surnames via `generate_last_names`. Needs no key; every element gets a new random surname, so repeated names are not mapped consistently.

Value

Character vector of pseudonyms matching the length of `name`; `NA` inputs stay `NA`.

See Also

`pi_anonymizer` for scope, limits and key setup.

Examples

```
# The key comes from the TEMPLECBE_SECRET_KEY environment variable. In real use
# keep it in your user ~/.Renviron (see ?pi_anonymizer), never in a script you
# commit. Here a throwaway value is set for this example only.
if (requireNamespace("withr", quietly = TRUE)) {
  withr::with_envvar(c(TEMPLECBE_SECRET_KEY = "example-only-not-a-real-key"), {
    # Vectorized deterministic tokens (default method = "hmac_token")
    print(anonymize_pi(c("Franklin", "Taylor", "Patel")))

    # Short tokens (8 characters)
```

```
    print(anonymize_pi(c("Franklin", "Taylor"), n_chars = 8))

    # Custom renaming function
    print(anonymize_pi(c("Franklin", "Taylor"), rename_fn = tolower))

    # Vectorized realistic pseudonyms
    print(anonymize_pi(c("Franklin", "Taylor"), method = "hmac_surname"))
  })
}

# Random tokens kept in a mapping file need no key. Here the file goes in a
# temporary directory; in real use leave secrets_path at its default.
mapping <- file.path(tempdir(), "pi_mapping_example.json")
anonymize_pi(c("Franklin", "Taylor", "Franklin"), method = "token", secrets_path = mapping)
unlink(mapping)
```

`apply_database_metadata` *Apply Metadata and Variable Roles to an R Database*

Description

Synchronizes updated metadata (such as variable labels, dataset descriptions, database names, and variable roles) back into a database list or data frame.

Usage

```
apply_database_metadata(db, metadata)
```

Arguments

`db`	A named list of data frames (an R database) or a single data frame.
`metadata`	A data frame/tibble of metadata containing columns `columns` and optionally `dataset_name`, `labels`, `dataset_label`, `database_name`, `role`, `X_var`, `Y_var`, `ID_var`, `Time_var`.

Value

The modified database or data frame with attributes and labels updated.

Examples

```
db <- list(demo = data.frame(id = 1:2, age = c(20, 30)))
meta <- get_dataset_info(db)
meta$labels[meta$columns == "age"] <- "Age at Enrollment (years)"
db <- apply_database_metadata(db, meta)
```

as_database *Convert a Graph Object Back Into an R Database*

Description

Reverses the data.base -> graph_object direction already covered by as_igraph (and cbe_database_relationship extracts a graph's node and edge tables into a named list of data frames (an R database), the same list(nodes = ..., edges = ...) shape used throughout this package's data.base <-> excel tooling (see write_workbook/read_workbook).

Usage

```
as_database(x, ...)

## Default S3 method:
as_database(x, ...)

## S3 method for class 'igraph'
as_database(x, ...)

## S3 method for class 'visNetwork'
as_database(x, ...)

## S3 method for class 'list'
as_database(x, ...)
```

Arguments

x	A graph object: an igraph (or tbl_graph, which extends it and is dispatched via the igraph method), a **visNetwork** htmlwidget (e.g. from visNetwork::visNetwork() or funviewR::plot_dependency_graph()), or a plain list(nodes = ..., edges = ...) (e.g. read back from read_workbook) to be validated and stamped as one.
...	Additional arguments passed to methods.

Details

Every node/edge attribute present on x is preserved as its own column – nothing is dropped, renamed, or flattened (e.g. a rich FilePath-derived pipeline graph's stage/mtime/ description/artifact_role, see compute_graph, all come through intact). There's no merging step here since only one graph is involved; the "whose attributes win" question only arises once you combine two databases – see database_setops for that.

Value

A named list list(nodes = <tibble>, edges = <tibble>) of class cbe_database – the class database_setops and cbe_compare_df dispatch on.

Examples

```
g <- igraph::make_ring(5)
as_database(g)
```

as_igraph.cbe_database_relationships *Convert to an enriched igraph graph with rich metadata*

Description

Convert to an enriched igraph graph with rich metadata

Usage

```
## S3 method for class 'cbe_database_relationships'
as_igraph(x, ...)

as_igraph(x, ...)

## Default S3 method:
as_igraph(x, stage_colors = NULL, ...)
```

Arguments

x	list of FilePath, FileUses, and FileOutputs objects, or another object with an as_igraph method (e.g. cbe_database_relationships).
...	Additional arguments passed to methods.
stage_colors	Optional named list of stage colors. If NULL, uses pipeline_config() options.

Details

A node is identified by its name, not by its path. Two objects of the list with the same name are an error ("Duplicate pipeline object name(s) detected"); a name that comes back only as a copy inside a dependencies or output list is one node. A file declared under two names is two nodes, so staleness does not pass from one to the other, and one name used for two files is one node. Both are reported with a warning, once per call, and so are two producers that declare the same output file. get_render_plan, visualize_pipeline and the summaries built on this function make the same checks.

Value

an igraph object containing complete node and edge attributes

as_pipeline_graph *Convert pipeline objects or igraph to a tidygraph tbl_graph*

Description

Enables native dplyr verbs (activate, filter, mutate, group_by, summarise) on computational graphs.

Usage

```
as_pipeline_graph(x, ...)
```

Arguments

x	A list of FilePath/FileUses/FileOutputs objects, or an igraph object.
...	Additional arguments passed to tidygraph::as_tbl_graph.

Value

A tbl_graph object.

---

as_tbl_graph.list *Convert pipeline list to tidygraph*

Description

A list of pipeline objects is converted by [as_pipeline_graph()]; any other list is passed on to tidygraph's own list method, which this method would otherwise replace.

Usage

```
## S3 method for class 'list'
as_tbl_graph(x, ...)
```

Arguments

x	A list of FilePath/FileUses/FileOutputs objects.
...	Additional arguments passed to as_pipeline_graph.

Value

A tbl_graph object.

---

audit_report_deliverables *Audit Deliverable File Tokens in Analysis Scripts*

Description

Scans source Quarto and R scripts to extract read/write file references and verify whether declared data and report deliverables exist on disk.

Usage

```
audit_report_deliverables(
  analysis_path = ".",
  file_pattern = "\\.(xlsx|rds|pdf|docx|html)$"
)
```

Arguments

analysis_path	Directory containing Quarto (.qmd) and R (.R) scripts.
file_pattern	Regular expression matching deliverable extensions (default: "\.(xlsx|rds|pdf|docx|html)$").

Value

A tibble summarizing referenced deliverable files, source lines, and on-disk existence.

---

autoplot.cbe_compare_df

Autoplot Method for CBE Data Frame Comparison

Description

Generates ggplot2 visualizations of data frame comparisons: 2-set Venn diagrams of observation/key overlap or variable concordance (leveraging **ggVennDiagram** when installed), or discrepancy bar charts across variables.

Usage

```
## S3 method for class 'cbe_compare_df'
autoplot(object, type = c("observations", "variables", "discrepancies"), ...)
```

Arguments

`object`	A `cbe_compare_df` object.
`type`	Character string specifying the plot type: `"observations"` (default; Venn of rows/keys), `"variables"` (Venn of common/unique columns), or `"discrepancies"` (bar chart of value differences).
`...`	Additional arguments passed to methods or **ggVennDiagram**.

Value

A `ggplot` object.

---

autoplot.cbe_database_relationships

Autoplot Method for Database Relationships

Description

Autoplot Method for Database Relationships

Usage

```
## S3 method for class 'cbe_database_relationships'
autoplot(object, type = c("graph", "venn"), db = NULL, id_col = NULL, ...)
```

Arguments

`object`	A `cbe_database_relationships` object.
`type`	Plot type: `"graph"` for network ER graph, or `"venn"` for key overlap.
`db`	Optional parent database list required when `type = "venn"`.
`id_col`	Identifier column for Venn overlap (default: first shared key).
`...`	Additional arguments passed to plotting backends.

Value

A ggplot2 or plot object.

`autoplot.cbe_key_integrity`
Autoplot Method for Key Integrity

Description

Autoplot Method for Key Integrity

Usage

```
## S3 method for class 'cbe_key_integrity'
autoplot(object, ...)
```

Arguments

`object`	A `cbe_check_key_integrity` object.
`...`	Additional arguments passed to methods.

Value

A ggplot2 object.

`autoplot.cbe_loco_mp_coxnet`
Autoplot Method for LOCO-MP Feature Importance

Description

Generates a ggplot2 forest/lollipop chart displaying Leave-One-Covariate-Out importance scores, confidence intervals, and significance thresholds.

Usage

```
## S3 method for class 'cbe_loco_mp_coxnet'
autoplot(object, ...)
```

Arguments

`object`	A `cbe_loco_mp_coxnet` object.
`...`	Additional arguments.

Details

The bars are the two-sided `1 - alpha` confidence intervals, which are not adjusted for multiple testing, but a predictor is coloured "Significant" when its adjusted *one-sided* p-value is below `alpha`, not when its bar excludes 0. The two criteria can disagree in either direction: with `p_adjust = "none"`, a "Significant" predictor whose p-value is between `alpha / 2` and `alpha` has a bar that still crosses 0, and after a correction such as Bonferroni a bar that excludes 0 can be "Not Significant".

Value

A ggplot2 plot object.

`autoplot.cv_joint_model`

Autoplot Method for Cross-Validated Joint Models

Description

Visualizes comparative predictive performance across paradigms, plotting the distribution of Integrated Brier Scores and Concordance.

Usage

```
## S3 method for class 'cv_joint_model'
autoplot(object, ...)
```

Arguments

`object`	A `cv_joint_model` object.
`...`	Additional arguments.

Value

A ggplot2 object.

`build_manual_versioned`

Build a Versioned PDF Reference Manual

Description

Renders the package's Rd documentation to a PDF manual (via `R CMD Rd2pdf`) named with the current package version, so successive builds never collide or overwrite each other. Writes into `pkgdown/assets/manual/` by default: pkgdown copies `pkgdown/assets/*` verbatim into `docs/` on every `pkgdown::build_site()`, which is what makes a navbar link to the manual resolve on the published site. (`inst/manual/` would not be copied there, and would also ship with every installed copy of the package.)

Usage

```
build_manual_versioned(pkg = ".", path = NULL, latest = TRUE)
```

Arguments

`pkg`	Path to the package, passed to `as.package`.
`path`	Directory to write the PDF into. Defaults to `pkgdown/assets/manual/` under the package root.
`latest`	Logical (default `TRUE`); if `TRUE`, also writes a stable `<package>_latest.pdf` copy alongside the versioned file, so a permanent link (e.g. in a pkgdown navbar) doesn't need to change on every version bump.

Value

The path to the versioned PDF, invisibly.

Examples

```
## Not run:
build_manual_versioned()

## End(Not run)
```

---

calculate_fences *Calculate Inner and Outer IQR Fences*

Description

Calculate Inner and Outer IQR Fences

Usage

```
calculate_fences(col)
```

Arguments

col	A numeric vector or column.

Value

A one-row tibble with columns `lower_inner_fence`, `upper_inner_fence` (at 1.5 x IQR — conventional "mild" outlier boundary) and `lower_outer_fence`, `upper_outer_fence` (at 3 x IQR — "extreme" outlier boundary).

Examples

```
calculate_fences(c(1, 2, 3, 4, 5, 100))
```

---

cbe_balloon_plot *Balloon Plot for 2-Way Contingency Tables*

Description

Dedicated wrapper for `cbe_contingency_plot` with `type = "balloon"`.

Usage

```
cbe_balloon_plot(
  data,
  var1 = NULL,
  var2 = NULL,
  label1 = NULL,
  label2 = NULL,
  engine = c("ggplot", "corrplot", "ggpubr"),
  size_range = c(6, 22),
  fill_colors = c("#F4F4F4", "#9D2235"),
  show_counts = TRUE,
  title = NULL,
  ...
)
```

Arguments

data	A data frame containing the categorical variables, or a 2-way table/matrix.
var1	Character string of the row variable column name (ignored if data is a table).
var2	Character string of the column variable column name (ignored if data is a table).
label1	Optional display label for var1 (defaults to var1).
label2	Optional display label for var2 (defaults to var2).
engine	Engine for balloon/square rendering: "ggplot" (default), "corrplot", or "ggpubr".
size_range	Numeric vector of length 2 giving min and max glyph sizes for balloon and square plots (default c(6, 22)).
fill_colors	Character vector of colors for the palette or gradient. Defaults to Temple University brand palette (c("#F4F4F4", "#9D2235")).
show_counts	Logical (default TRUE); whether to print cell counts inside plots.
title	Plot title. If NULL, defaults to "{label1} \u00d7 {label2}".
...	Additional arguments passed to underlying plotting functions.

Value

A ggplot object or invisibly the contingency table.

---

cbe_bar_plot *Bar Chart for 2-Way Contingency Tables*

Description

Dedicated wrapper for cbe_contingency_plot with type = "bar".

Usage

```
cbe_bar_plot(
  data,
  var1 = NULL,
  var2 = NULL,
  label1 = NULL,
  label2 = NULL,
  bar_position = c("fill", "dodge", "stack"),
  show_counts = TRUE,
  title = NULL,
  ...
)
```

Arguments

data	A data frame containing the categorical variables, or a 2-way table/matrix.
var1	Character string of the row variable column name (ignored if data is a table).
var2	Character string of the column variable column name (ignored if data is a table).
label1	Optional display label for var1 (defaults to var1).
label2	Optional display label for var2 (defaults to var2).
bar_position	Position adjustment when type = "bar": "fill" (100% proportional stacked bar, default), "dodge" (side-by-side grouped), or "stack" (count stacked).
show_counts	Logical (default TRUE); whether to print cell counts inside plots.
title	Plot title. If NULL, defaults to "{label1} \u00d7 {label2}".
...	Additional arguments passed to underlying plotting functions.

Value

A ggplot object.

---

cbe_compare_df *Compare Two Data Frames (Modelled on SAS PROC COMPARE)*

Description

Compares two data frames (or tibbles) at the dataset, variable, and observation levels, providing an audit-ready, tidy alternative to SAS PROC COMPARE and legacy comparison utilities (such as arsenal::comparedf). Evaluates variable presence, data types, variable labels (from **labelled**), row counts, and value-level discrepancies within a user-specified numerical tolerance. It agrees with PROC COMPARE on the benchmark described under "Agreement with SAS PROC COMPARE", and differs from it in the ways listed there.

Usage

```
cbe_compare_df(base, compare, ...)

## Default S3 method:
cbe_compare_df(
  base,
  compare,
  by = NULL,
  tolerance = 1e-07,
  base_name = NULL,
  compare_name = NULL,
  max_diffs = 100,
  ...
)

## S3 method for class 'cbe_database'
cbe_compare_df(
  base,
  compare,
  by = NULL,
  edge_by = c("from", "to"),
  ...,
  base_name = NULL,
  compare_name = NULL
)

## S3 method for class 'igraph'
cbe_compare_df(base, compare, ...)

## S3 method for class 'visNetwork'
cbe_compare_df(base, compare, ...)

compare_df(base, compare, ...)
```

Arguments

base	The base data frame or tibble (equivalent to SAS base=).
compare	The comparison data frame or tibble (equivalent to SAS compare=).
...	Passed on to methods (and, for the cbe_database method, on to the per-table cbe_compare_df() calls – e.g. by/tolerance).
by	Optional character vector of column names specifying key variables used to align observations across datasets (equivalent to SAS id or by statement). If NULL (default), observations are compared by row order. Rows that share a key are paired in order of appearance (with a warning); surplus duplicates on either side count as unmatched. A key column named like one of the diffs columns (variable, label, ...) is prefixed with key_ there. For the cbe_database method, this is the node key (passed through to the nodes comparison only).
tolerance	Non-negative numeric threshold for numeric differences (default: 1e-7). Differences with absolute magnitude less than or equal to tolerance are considered matches.

base_name	Optional character string identifying the base dataset. If NULL, deparsed from the base argument.
compare_name	Optional character string identifying the comparison dataset. If NULL, deparsed from the compare argument.
max_diffs	Maximum number of discrepant rows to store per variable (default: 100).
edge_by	cbe_database method only: key column(s) used to align edges (default c("from", "to"), which every as_database method produces).

Value

An S3 object of class "cbe_compare_df" containing (or, for a cbe_database/graph comparison, a "cbe_compare_database" list of two such objects, named nodes and edges):

meta	List containing metadata on dataset names, dimensions, tolerance, and keys.
variables	List with elements common, base_only, and compare_only.
summary	Tibble summarizing comparison status for each variable (types, match status, difference counts, max difference, RMSE).
observations	List summarizing observation matching (counts, keys matched, unmatched).
diffs	Tibble containing detailed cell-by-cell discrepancies for all variables exceeding tolerance.
is_concordant	Logical indicating whether the datasets match completely within tolerance on all common variables and observations.

Agreement with SAS PROC COMPARE

The package tests compare the results with SAS 9.4 PROC COMPARE (METHOD=ABSOLUTE, CRITERION equal to tolerance, by as the ID statement) on nine small comparisons typed into 'inst/sas/benchmark_proc_compare its listing is committed under 'inst/sas/list/'. On that data both agree on the number of variables in common and only in either data set; of observations matched and only in either data set (by key, by position, and with repeated keys paired in order); of values that differ, per variable and in all, and of observations with a difference; of values that are not exactly equal (tolerance = 0); on the maximum difference; and on every cell that differs. A value differs if abs(difference) > tolerance (a difference of exactly tolerance is none), and a missing value against a value is a difference. The agreement is on the variables SAS compares (see the second item below) and on that data; it is not a proof for other inputs.

The two define some things differently:

- The sign: diff in diffs is base minus compare; PROC COMPARE prints Diff. as compare minus base.
- A variable whose type differs between the data sets is still compared, as text, and reported with types_match = FALSE; PROC COMPARE compares nothing and counts a conflicting type.
- "Type" is the R class: integer against numeric, or factor against character, is a type mismatch, although SAS has only numeric and character variables.
- Strings are compared as they are: a trailing blank is a difference (SAS pads the shorter value), and NA differs from "" (SAS's missing character value is the blank).
- Only the type is checked, not the length, format, informat or label (SAS reports a character variable of different lengths as a differing attribute).
- Only absolute differences. METHOD=PERCENT and RELATIVE, BY group reports and special missing values are not part of the benchmark.
- cbe_compare_df() needs no sorted data; PROC COMPARE does (the benchmark sorts the COMPARE data set).

Examples

```
df1 <- data.frame(id = 1:5, x = c(1, 2, 3, 4, 5), y = c("a", "b", "c", "d", "e"))
df2 <- data.frame(id = 1:5, x = c(1, 2, 3.0001, 4, 5), y = c("a", "b", "c", "d", "f"))
cmp <- cbe_compare_df(df1, df2, by = "id", tolerance = 1e-3)
print(cmp)
generics::tidy(cmp)
g1 <- igraph::graph_from_data_frame(
  data.frame(from = c("a", "b"), to = c("b", "c")),
  vertices = data.frame(name = c("a", "b", "c"), stage = c("eda", "eda", "report"))
)
g2 <- igraph::graph_from_data_frame(
  data.frame(from = c("a", "b"), to = c("b", "c")),
  vertices = data.frame(name = c("a", "b", "c"), stage = c("eda", "analysis", "report"))
)
cbe_compare_df(g1, g2, by = "name")
```

---

cbe_contingency_plot *Visualizations for 2-Way Contingency Tables and Categorical Tests*

Description

Generates publication-ready clinical visualizations for 2-way contingency tables with hypothesis test statistics (Central Fisher's Exact Test with mid-p adjustment when zero-cells occur, or Pearson Chi-squared test) automatically formatted in the caption.

Usage

```
cbe_contingency_plot(
  data,
  var1 = NULL,
  var2 = NULL,
  label1 = NULL,
  label2 = NULL,
  type = c("balloon", "bar", "mosaic", "heatmap", "square", "corrplot"),
  bar_position = c("fill", "dodge", "stack"),
  engine = c("ggplot", "corrplot", "ggpubr"),
  test = c("auto", "exact", "chisq", "fisher"),
  correct = FALSE,
  size_range = c(6, 22),
  fill_colors = c("#F4F4F4", "#9D2235"),
  show_counts = TRUE,
  title = NULL,
  caption = NULL,
  p_format = NULL,
  ...
)

contingency_plot(
  data,
  var1 = NULL,
  var2 = NULL,
  label1 = NULL,
```

```
  label2 = NULL,
  type = c("balloon", "bar", "mosaic", "heatmap", "square", "corrplot"),
  bar_position = c("fill", "dodge", "stack"),
  engine = c("ggplot", "corrplot", "ggpubr"),
  test = c("auto", "exact", "chisq", "fisher"),
  correct = FALSE,
  size_range = c(6, 22),
  fill_colors = c("#F4F4F4", "#9D2235"),
  show_counts = TRUE,
  title = NULL,
  caption = NULL,
  p_format = NULL,
  ...
)

balloon_plot(
  data,
  var1 = NULL,
  var2 = NULL,
  label1 = NULL,
  label2 = NULL,
  engine = c("ggplot", "corrplot", "ggpubr"),
  size_range = c(6, 22),
  fill_colors = c("#F4F4F4", "#9D2235"),
  show_counts = TRUE,
  title = NULL,
  ...
)

bar_plot(
  data,
  var1 = NULL,
  var2 = NULL,
  label1 = NULL,
  label2 = NULL,
  bar_position = c("fill", "dodge", "stack"),
  show_counts = TRUE,
  title = NULL,
  ...
)

mosaic_plot(
  data,
  var1 = NULL,
  var2 = NULL,
  label1 = NULL,
  label2 = NULL,
  show_counts = TRUE,
  title = NULL,
  ...
)
```

```
heatmap_plot(
  data,
  var1 = NULL,
  var2 = NULL,
  label1 = NULL,
  label2 = NULL,
  fill_colors = c("#F4F4F4", "#9D2235"),
  show_counts = TRUE,
  title = NULL,
  ...
)

four_quadrant_report(
  data,
  var1 = NULL,
  var2 = NULL,
  label1 = NULL,
  label2 = NULL,
  test = c("auto", "exact", "chisq", "fisher"),
  correct = FALSE,
  p_format = NULL,
  ...
)

square_plot(
  data,
  var1 = NULL,
  var2 = NULL,
  label1 = NULL,
  label2 = NULL,
  test = c("auto", "exact", "chisq", "fisher"),
  correct = FALSE,
  engine = c("ggplot", "corrplot"),
  size_range = c(6, 22),
  fill_colors = c("#F4F4F4", "#9D2235"),
  show_counts = TRUE,
  title = NULL,
  caption = NULL,
  p_format = NULL,
  ...
)
```

Arguments

data	A data frame containing the categorical variables, or a 2-way table/matrix.
var1	Character string of the row variable column name (ignored if data is a table).
var2	Character string of the column variable column name (ignored if data is a table).
label1	Optional display label for var1 (defaults to var1).
label2	Optional display label for var2 (defaults to var2).
type	Visualization type: "balloon" (default), "bar", "mosaic", "heatmap", "square", or "corrplot".

bar_position	Position adjustment when type = "bar": "fill" (100% proportional stacked bar, default), "dodge" (side-by-side grouped), or "stack" (count stacked).
engine	Engine for balloon/square rendering: "ggplot" (default), "corrplot", or "ggpubr".
test	Hypothesis test engine: "auto" (default CBE hierarchy), "exact" (force Central Fisher exact test for 2x2 or simulated Fisher for RxC), "chisq" (force Pearson's Chi-squared test via chisq.test), or "fisher" (force fisher.test).
correct	Logical; whether to apply continuity correction when test = "chisq" (default FALSE following CBE standard protocol).
size_range	Numeric vector of length 2 giving min and max glyph sizes for balloon and square plots (default c(6, 22)).
fill_colors	Character vector of colors for the palette or gradient. Defaults to Temple University brand palette (c("#F4F4F4", "#9D2235")).
show_counts	Logical (default TRUE); whether to print cell counts inside plots.
title	Plot title. If NULL, defaults to "{label1} \u00d7 {label2}".
caption	Optional plot caption. If NULL, automatically computed and formatted using pformat with test description.
p_format	Optional p-value formatting function (default pformat).
...	Additional arguments passed to underlying plotting functions.

Details

Supported visualization types:

- "balloon" (default): Proportional bubble/circle plot with counts displayed inside circles.
- "bar": Grouped, stacked, or 100% proportional fill bar chart.
- "mosaic": Two-dimensional mosaic plot with tile area proportional to joint frequency.
- "heatmap": Shaded tile heatmap with cell frequencies and overall percentages.
- "square": Standard 4-quadrant report for 2x2 tables (q1 | q2 // q3 | q4 // p = pformat), or proportional square tiles for RxC tables.
- "corrplot": Circular or square matrix plot rendered via **corrplot**.

Value

A ggplot object (for "balloon", "bar", "mosaic", "heatmap", "square"), or invisibly the contingency table (for "corrplot").

Examples

```
df <- data.frame(
  Treatment = factor(c(rep("Active", 20), rep("Control", 20))),
  Outcome   = factor(c(rep("Response", 14), rep("None", 6),
                       rep("Response", 5), rep("None", 15)))
)

# 1. Balloon Plot (Temple Brand Colors)
cbe_contingency_plot(df, "Outcome", "Treatment", type = "balloon")

# 2. Proportional Bar Plot (100% Fill)
cbe_contingency_plot(df, "Outcome", "Treatment", type = "bar", bar_position = "fill")
```

```
# 3. Mosaic Plot
cbe_contingency_plot(df, "Outcome", "Treatment", type = "mosaic")

# 4. Heatmap
cbe_contingency_plot(df, "Outcome", "Treatment", type = "heatmap")

# 5. Standard 4-Quadrant Square Plot (q1 | q2 // q3 | q4 // p = pformat)
cbe_contingency_plot(df, "Outcome", "Treatment", type = "square")

# 6. Force Pearson Chi-Square Test
cbe_contingency_plot(df, "Outcome", "Treatment", type = "square", test = "chisq")
```

---

cbe_cox_check *Tidy Proportional Hazards Diagnostics for Cox Models*

Description

Standalone proportional hazards (PH) assumption diagnostics, usable on any fitted `survival::coxph` model, or on a `cbe_cox` or `cbe_cox_multi` object. Wraps `survival::cox.zph()` with a tidy per-term table, a per-term violation flag, and an automated text summary.

Usage

```
cbe_cox_check(fit)

cox_check(fit)
```

Arguments

`fit`	A fitted `survival::coxph` model, or a `cbe_cox` or `cbe_cox_multi` object.

Value

An object of class `cbe_cox_check` containing:

- `zph`: The `survival::cox.zph` object (or `NULL` if it could not be computed).
- `zph_table`: Data frame version of `zph$table` (one row per term, plus a `GLOBAL` row for multivariable fits).
- `zph_violated`: Named logical vector, one entry per term (excluding `GLOBAL`), `TRUE` where the assumption is violated ($p < 0.05$). For a univariable fit this is a length-1 named logical vector.
- `zph_text`: Automated summary sentence (a single string for a univariable fit with one term; a two-element character vector, per-term then global, for a multivariable fit).

See Also

[cbe_cox_single()], [cbe_cox_multi()]

cbe_cox_multi *Multivariable Cox Proportional Hazards Modeling Engine*

Description

Fits a multivariable Cox proportional hazards model, validates the proportional hazards assumption for every term (including the multivariate global test), formats coefficients with explicit reference levels grouped by variable, and gathers model fit and convergence metrics.

Usage

```
cbe_cox_multi(
  data,
  formula = NULL,
  outcome = "outcome",
  features = NULL,
  conf_level = 0.95,
  ...
)

cox_multi(
  data,
  formula = NULL,
  outcome = "outcome",
  features = NULL,
  conf_level = 0.95,
  ...
)
```

Arguments

data	A data frame containing survival data and the candidate features.
formula	Optional. A model formula with a survival::Surv() outcome, e.g. Surv(time, status) ~ age + sex. When supplied, outcome and features are ignored.
outcome	Character string naming the Surv object or outcome column in data (e.g., "outcome" or "Surv(survival_time, status)"). Ignored if formula is supplied.
features	Character vector of candidate predictor columns in data. Required if formula is not supplied.
conf_level	Numeric confidence level (default: 0.95).
...	Additional arguments passed to survival::coxph (e.g., id, ties, weights).

Value

An object of class cbe_cox_multi containing:

- model: The fitted survival::coxph object.
- table: Tidy coefficient table (HR, log-HR, CI, p-value) with explicit reference rows per factor, grouped by Variable.
- glance: One-row data frame of model goodness-of-fit metrics.

- zph: A cbe_cox_check object (includes the multivariate global test).
- converged: Logical; whether the fit converged within the iteration limit.
- n_iterations: Number of Newton-Raphson iterations used by the fit.
- features: Character vector of feature names used.
- var_labels: Named character vector mapping feature names to labels.

See Also

[cbe_cox_single()], [cbe_cox_check()], [cbe_cox_table()], [plot_cox_forest_multi()]

---

cbe_cox_single *Univariable Cox Proportional Hazards Screening Engine*

Description

Fits a univariable Cox proportional hazards model, validates the proportional hazards assumption, formats coefficients with explicit reference levels, generates plain-language clinical interpretations, and gathers model fit metrics.

Usage

```
cbe_cox_single(data, outcome = "outcome", feature, conf_level = 0.95, ...)

cox_single(data, outcome = "outcome", feature, conf_level = 0.95, ...)
```

Arguments

data	A data frame containing survival data and the candidate feature.
outcome	Character string naming the Surv object or outcome column in data (e.g., "outcome" or "Surv(survival_time, status)").
feature	Character string naming the candidate predictor column in data.
conf_level	Numeric confidence level (default: 0.95).
...	Additional arguments passed to survival::coxph (e.g., weights, ties).

Value

An object of class cbe_cox containing:

- model: The fitted survival::coxph object.
- zph: The survival::cox.zph proportional hazards test object.
- zph_table: Formatted assumption test table.
- zph_violated: Logical indicating whether the assumption was violated ($p < 0.05$).
- zph_text: Automated sentence summarizing the assumption test.
- table: Clean coefficient table with explicit reference level rows for factors.
- interpretation: Plain-language automated interpretation of the hazard ratio(s).
- glance: One-row data frame of model goodness-of-fit metrics.
- feature: Feature name string.
- var_label: Human-readable variable label.
- is_numeric: Logical flag indicating whether predictor is continuous/numeric.

See Also

[cbe_cox_multi()], [cbe_cox_check()], [cbe_km_single()], [cbe_cox_table()], [plot_cox_forest()], [plot_cox_survival()], [plot_cox_marginal()]

`cbe_cox_table` *Presentation-Ready Cox Coefficient Table*

Description

Formats the `table` element of a [cbe_cox_single()] or [cbe_cox_multi()] result into a presentation table, adding a log(HR) column (if not already present) and optional sorting and significance annotation.

Usage

```
cbe_cox_table(x, sort = c("none", "magnitude", "pvalue"), significance = FALSE)

cox_table(x, sort = c("none", "magnitude", "pvalue"), significance = FALSE)
```

Arguments

`x`	A `cbe_cox` (from [cbe_cox_single()]) or `cbe_cox_multi` (from [cbe_cox_multi()]) object.
`sort`	One of `"none"` (default; original model order), `"magnitude"` (sorts by `abs(log(HR))` descending; reference rows, which are always exactly 0, sort last), or `"pvalue"` (sorts by p-value ascending; reference rows, which have no p-value, sort last).
`significance`	Logical; if `TRUE`, appends a `sig` column of significance stars (three asterisks for $p < 0.001$, two for $p < 0.01$, one for $p < 0.05$) and bolds the `p.value` text of significant rows.

Value

A tibble formatted for presentation, with columns `Variable`, `Level`, `Role`, `HR`, `log(HR)`, `95% CI`, `p.value`, and (if `significance = TRUE`) `sig`.

See Also

[cbe_cox_single()], [cbe_cox_multi()]

`cbe_database_relationships` *Analyze Relationships and Linkages Across Datasets in an R Database*

Description

Evaluates common key variables (e.g., subject identifiers, visit times) across multiple datasets in an R database list. Computes cross-table overlap, record coverage, key cardinality (1:1, 1:many, many:many), and identifies orphan records.

Usage

```
cbe_database_relationships(db, id_cols = NULL)

## S3 method for class 'cbe_database_relationships'
as_tbl_graph(x, ...)

cbe_find_shared_keys(db)

cbe_check_key_integrity(
  db,
  id_col,
  master_dataset = NULL,
  compare = FALSE,
  tolerance = 1e-07,
  ...
)

database_relationships(db, id_cols = NULL)

find_shared_keys(db)

check_key_integrity(
  db,
  id_col,
  master_dataset = NULL,
  compare = FALSE,
  tolerance = 1e-07,
  ...
)
```

Arguments

`db`	A named list of data frames (an R database).
`id_cols`	Optional character vector of key/ID column names to evaluate. If `NULL`, shared keys are automatically detected across datasets.
`x`	A `cbe_database_relationships` object.
`...`	Additional arguments passed to `cbe_compare_df`.
`id_col`	Character string specifying the identifier column to check.

master_dataset	Character string naming the primary cohort table. If provided, checks which IDs in other tables are absent from the master cohort table.
compare	Logical; if TRUE and master_dataset is supplied, runs cbe_compare_df on overlapping variables between the master dataset and child datasets, attaching the comparison results.
tolerance	Numeric comparison tolerance passed to cbe_compare_df.

Value

A tibble summarizing dataset pairs, shared key columns, record counts, overlap counts, coverage percentages, and cardinality.

Examples

```
db <- list(
  patients = data.frame(id = 1:5, age = c(20, 30, 40, 50, 60)),
  vitals = data.frame(id = c(1, 1, 2, 2, 3, 6), bp = c(120, 122, 130, 132, 110, 115))
)
cbe_database_relationships(db)
```

---

cbe_database_venn *Multi-Dataset Key Overlap Venn Diagram*

Description

Visualizes shared subject identifiers or key overlap across all tables in an R database list using **ggVennDiagram** (or native ggplot2 fallback).

Usage

```
cbe_database_venn(db, id_col = "id", title = NULL, ...)

database_venn(db, id_col = "id", title = NULL, ...)
```

Arguments

db	A named list of data frames (an R database).
id_col	Character string specifying the identifier column to intersect across datasets.
title	Optional plot title.
...	Additional arguments passed to ggVennDiagram.

Value

A ggplot object.

Examples

```
## Not run:
db <- list(
  inputs = data.frame(id = 1:5),
  abg = data.frame(id = 1:3),
  survival = data.frame(id = 2:6)
)
cbe_database_venn(db, id_col = "id")

## End(Not run)
```

`cbe_docx_review_extract` *DOCX Review Extractor & Multi-Reviewer Collaborative Tracker*

Description

Extracts comments, tracked changes, and reconstructed paragraph redlines directly from Microsoft Word (‘.docx‘) files and maintains a collaborative review workflow in a formatted multi-sheet Excel workbook and UTF-8 CSV files.

Supports multi-reviewer workflows with per-reviewer fork files (‘review_tracker_<reviewer>.xlsx‘), live Excel formulas, dropdown data validations, conditional formatting, non-destructive merging across document revisions, and analytical compute graph ordering.

Usage

```
cbe_docx_review_extract(
  input_dir = NULL,
  output_dir = NULL,
  tracker = "review_tracker.xlsx",
  min_revision_length = 2,
  fork_reviewers = NULL,
  signoff_reviewers = NULL,
  documents_sheet_mode = NULL,
  manifest_path = NULL,
  verbose = FALSE,
  docx_limits = NULL
)

docx_review_extract(
  input_dir = NULL,
  output_dir = NULL,
  tracker = "review_tracker.xlsx",
  min_revision_length = 2,
  fork_reviewers = NULL,
  signoff_reviewers = NULL,
  documents_sheet_mode = NULL,
  manifest_path = NULL,
  verbose = FALSE,
  docx_limits = NULL
)
```

Arguments

input_dir	Character path to a directory containing ‘.docx‘ files, or to a single ‘.docx‘ file. Defaults to the first folder of ‘review_config()$input_dirs‘ that holds ‘.docx‘ files (none is configured by default), else the current working directory.
output_dir	Character path to the directory where review tracker workbooks and CSV files should be saved. If ‘NULL‘ (default), uses ‘<input_dir>/review_extract‘ (or ‘<input_dir>‘ if it is already named ‘review_extract‘).
tracker	Character filename for the master Excel workbook. Defaults to ‘"review_tracker.xlsx"‘.
min_revision_length	Integer minimum character count required to record a tracked change. Defaults to ‘2‘.
fork_reviewers	Character vector of reviewer identifiers who receive their own single-reviewer fork workbooks and whose per-comment sign-offs roll up into each document’s ‘resolved‘ status. Defaults to ‘review_config()$fork_reviewers‘ (generic placeholders unless a project has called review_config to configure real reviewer ids).
signoff_reviewers	Character vector of document-level sign-off reviewer identifiers (e.g. a final PI read), tracked on the master workbook’s "Documents" sheet and not counted toward ‘resolved‘; they appear in no fork workbook. Defaults to ‘review_config()$signoff_reviewers‘.
documents_sheet_mode	One of ‘"columns"‘ or ‘"per_reviewer"‘; see review_config. Defaults to ‘review_config()$documents_sheet_mode‘.
manifest_path	Optional character path to ‘reports_to_render.xlsx‘. If ‘NULL‘, the function looks for it in the folder set with ‘review_config(analysis_dir =)‘ (nothing is searched when none is set). Pipeline stages come from that manifest and from ‘review_config()$pipeline_catalog‘; with neither, documents are not assigned to a stage (stage ‘"Unknown / Extra"‘) and are listed by file name.
verbose	Logical indicating whether to print detailed progress messages. Defaults to ‘FALSE‘.
docx_limits	NULL (the defaults) or a named list limiting what is done with each .docx, which is an untrusted zip archive: max_entries (entries in the archive, default 10,000), max_total_bytes (summed uncompressed size, default 2 GiB) and max_xml_bytes (uncompressed size of any one XML part that is read, default 100 MiB). The listing is checked before anything is extracted, only the parts that are read are extracted, and a file over a limit is reported in $errors instead of being read. Inf switches a limit off. The session-wide options review.docx_max_entries, review.docx_max_bytes and review.docx_max_xml_bytes set the defaults.

Details

What is read. Comments (word/comments.xml, with reply threads and the resolved flag from word/commentsExtended.xml) and the tracked changes and comment ranges of the main document body (word/document.xml). A text box is read once and counts as part of the paragraph that holds it, so paragraph numbers are Word’s.

What is not read. Tracked changes and comment ranges in footnotes, endnotes, headers and footers are not extracted, and a comment anchored in one of them is listed with no paragraph number and no selected text. They are not dropped silently: when one of those parts holds either, a single warning names the files, the parts and how many there are, so that they can be reviewed in Word.

Merging a new round into the tracker. A comment is identified by what it says, not by its Word comment id (Word renumbers the ids when it saves). A comment continues an existing row of the same file only when its text is the same (compared without regard to whitespace); among rows with the same text the one with the same author and date is taken first, then one with the same author (or an author missing on one side), preferring the same comment id and then the nearest paragraph. A comment whose text changed is a new comment: the row it replaces is kept, marked `Prior Round / Not in docx`, when it holds a sign-off or a note. Rows of documents that left the folder, or that could not be read in this run, are kept the same way.

Reviewer forks. A reviewer's edits in their fork workbook (`review_tracker_<id>.xlsx`) are merged back by file and comment (or suggested change). Only cells the reviewer changed since the fork was written are taken (each fork records what it was written with on a hidden sheet), so a sign-off typed into the master workbook is not undone by a fork that left the cell alone. If both were changed to different values the fork's value is kept and a warning lists the cells; the previous master is in the `backups` folder.

Files that are not readable Word documents (empty, not a ZIP archive, no `word/document.xml`, or an XML error) are listed in `$errors` and on the Errors sheet and reported in one warning; Word's `~$` lock files are ignored. A run that finds no readable document creates and backs up nothing.

Value

An object of class 'cbe_review_extract', which is a named list containing:

comments A tibble of merged comment records.

revisions A tibble of merged raw tracked changes.

redlines A tibble of reconstructed paragraph before/after redlines.

documents A tibble of document-level summary metrics and review statuses.

docxwalk A tibble crosswalking reviewed DOCX files to source '.qmd' and rendered outputs.

summary A tibble summarizing recurring comments across documents.

errors A tibble of any processing errors encountered.

paths A named list of generated file paths.

Examples

```
## Not run:
# Configure real reviewer identities once per project (kept out of source
# control), then run extraction with no further reviewer arguments needed:
review_config(fork_reviewers = c("alice", "bob", "carol"), signoff_reviewers = "lead_pi")
# Optionally describe your own pipeline stages, so documents are ordered and
# labelled by stage (the package ships none):
review_config(pipeline_catalog = list(
  list(stem = "overview", heading = 1, name = "Overview"),
  list(stem = "model_fit", heading = 2, name = "Model fitting")
))
res <- cbe_docx_review_extract("path/to/reviewed_docx", verbose = TRUE)
print(res)

## End(Not run)
```

cbe_exact2x2 *Exact Test for 2x2 Tables with Automatic Zero-Cell Mid-p Default*

Description

Performs an exact test for 2x2 contingency tables using **exact2x2**. If any cell count in the 2x2 table is zero (`min(tab) == 0`) and `midp` is unspecified, the function defaults to the mid-p version of Central Fisher's exact test (`midp = TRUE`). This prevents the extreme conservatism and loss of power of standard conditional exact tests on boundary tables. When all cells are non-zero, it defaults to the standard Central Fisher's exact test (`midp = FALSE`), which guarantees central confidence intervals that invert the two one-sided tests.

Usage

```
cbe_exact2x2(
  x,
  y = NULL,
  data = NULL,
  variable = NULL,
  by = NULL,
  midp = NULL,
  conf.level = 0.95,
  alternative = "two.sided",
  tsmethod = "central",
  group = NULL,
  type = NULL,
  test.args = NULL,
  adj.vars = NULL,
  tbl = NULL,
  continuous_variable = NULL,
  ...
)

exact2x2(
  x,
  y = NULL,
  data = NULL,
  variable = NULL,
  by = NULL,
  midp = NULL,
  conf.level = 0.95,
  alternative = "two.sided",
  tsmethod = "central",
  group = NULL,
  type = NULL,
  test.args = NULL,
  adj.vars = NULL,
  tbl = NULL,
  continuous_variable = NULL,
  ...
)
```

Arguments

x	A 2x2 numeric matrix, table, or a categorical vector. Can also be a data frame if variable and by are supplied (by name).
y	Optional second categorical vector when x is a vector.
data	Optional data frame when used with variable and by.
variable	Character string of the column name to test when called by **gtsummary**.
by	Character string of the grouping/stratifying column when called by **gtsummary**.
midp	Logical. If NULL (the default), automatically set to TRUE when any table cell is 0, and FALSE when all cells are positive. Can be explicitly set to TRUE or FALSE to override.
conf.level	Confidence level for the returned confidence interval (default 0.95).
alternative	Alternative hypothesis direction: "two.sided" (default), "greater", or "less".
tsmethod	Two-sided method passed to exact2x2: "two.sided" (Central Fisher's exact test, default), "minlike", or "blaker".
group, type, test.args, adj.vars, tbl, continuous_variable	Unused; accepted and ignored so this function satisfies **gtsummary**'s custom-test calling convention (see tests) instead of leaking into ... and reaching exact2x2.
...	Additional arguments passed to exact2x2.

Details

The function accepts a 2x2 table/matrix, two categorical vectors x and y, or data, variable and by *by name* for direct compatibility as a custom test in **gtsummary**'s add_p. A table with an all-zero row or column (for example an unused factor level) is reduced to the observed levels first, so a binary variable with an empty extra level is still tested as 2x2.

Value

A tibble with columns:

estimate	Estimated odds ratio (conditional MLE or median unbiased estimate).
p.value	Two-sided or one-sided p-value (mid-p adjusted when midp = TRUE).
conf.low	Lower bound of the confidence interval.
conf.high	Upper bound of the confidence interval.
statistic	Point estimate of the odds ratio (for **gtsummary** compatibility).
method	Descriptive name of the exact test performed.
alternative	Alternative hypothesis.
midp	Logical flag indicating whether mid-p adjustment was used.
has_zero	Logical flag indicating whether any cell in the table was zero.

Examples

```
# 2x2 table with a zero cell: automatically triggers mid-p adjustment
tab_zero <- matrix(c(0, 10, 5, 15), nrow = 2,
                   dimnames = list(c("Treated", "Control"), c("Event", "No Event")))
cbe_exact2x2(tab_zero)
```

```
# 2x2 table without zero cells: standard Central Fisher's exact test
tab_nonzero <- matrix(c(12, 8, 5, 15), nrow = 2)
cbe_exact2x2(tab_nonzero)
```

`cbe_exact2x2_ci` *Format Exact 2x2 Odds Ratio and Confidence Interval*

Description

Computes the odds ratio and confidence interval from `cbe_exact2x2` and returns a formatted character string (e.g. `"0.8 (0.3, 2.1)"`). Useful for inline text reporting or table presentation.

Usage

```
cbe_exact2x2_ci(
  data,
  variable = NULL,
  by = NULL,
  digits = 1,
  conf.level = 0.95,
  midp = NULL,
  ...
)

exact2x2_ci(
  data,
  variable = NULL,
  by = NULL,
  digits = 1,
  conf.level = 0.95,
  midp = NULL,
  ...
)
```

Arguments

`data`	A data frame or a 2x2 table/matrix.
`variable`	Character string column name (if `data` is a data frame).
`by`	Character string grouping column name (if `data` is a data frame).
`digits`	Integer number of decimal places for rounding (default 1).
`conf.level`	Confidence level (default 0.95).
`midp`	Logical or `NULL` (default). If `NULL`, automatically defaults to `TRUE` if any cell count is zero.
`...`	Additional arguments passed to `cbe_exact2x2`.

Value

A character string with the formatted odds ratio and confidence interval.

Examples

```
tab <- matrix(c(0, 10, 5, 15), nrow = 2)
cbe_exact2x2_ci(tab)
```

`cbe_explain` *Explain Joint Model Components via DALEX and survex*

Description

Extracts and creates explainers for all components of a `joint_model`: the survival model (via `survex`), the binary status model (via `DALEX`), and the continuous duration time model (via `DALEX`).

Usage

```
cbe_explain(
  model,
  data = NULL,
  type = c("all", "survival", "status", "time", "stack"),
  times = NULL,
  label = NULL,
  verbose = FALSE,
  ...
)

cbe_explain_status(model, data = NULL, label = NULL, verbose = FALSE, ...)

cbe_explain_time(model, data = NULL, label = NULL, verbose = FALSE, ...)

cbe_explain_stack(model, data = NULL, label = NULL, verbose = FALSE, ...)

explain_joint(
  model,
  data = NULL,
  type = c("all", "survival", "status", "time", "stack"),
  times = NULL,
  label = NULL,
  verbose = FALSE,
  ...
)

explain_status(model, data = NULL, label = NULL, verbose = FALSE, ...)

explain_time(model, data = NULL, label = NULL, verbose = FALSE, ...)

explain_stack(model, data = NULL, label = NULL, verbose = FALSE, ...)
```

Arguments

`model`	A fitted `joint_model` object.

data	Evaluation data frame with the original predictor columns (before one-hot encoding; extra columns are ignored). If NULL, the training predictors are used, as the original columns. The status and time explainers re-apply the model's training encoding with forge, as predict.joint_model does, so a factor predictor is one column in the explainers too.
type	Character indicating which component(s) to explain: "all", "survival", "status", "time", or "stack".
times	Numeric vector of evaluation time points for survival explanations.
label	Optional custom label prefix.
verbose	Logical; if TRUE, progress messages are shown. Default is FALSE.
...	Additional arguments passed to the underlying explainers.

Value

If type = "all", a list of class "cbe_joint_explainer" containing $survival, $status, $time, and (if present) $stack explainers. Otherwise, the individual requested explainer.

---

cbe_explain_survival *Explain Survival Models via survex and DALEX*

Description

Adapts Tidymodels survival models (workflows, parsnip / censored, coxnet_model, and joint_model) to create a survex survival explainer. Standardizes survival predictions into $N \times K$ survival curves, bridging the gap identified in survex issue #98.

Usage

```
cbe_explain_survival(
  model,
  data = NULL,
  y = NULL,
  predict_survival_function = NULL,
  predict_risk_function = NULL,
  times = NULL,
  label = NULL,
  verbose = FALSE,
  ...
)

explain_survival(
  model,
  data = NULL,
  y = NULL,
  predict_survival_function = NULL,
  predict_risk_function = NULL,
  times = NULL,
  label = NULL,
  verbose = FALSE,
  ...
)
```

Arguments

model	A fitted model object: joint_model, coxnet_model, cv_coxnet, workflow, or model_fit.
data	A data frame containing predictor columns used to calculate explanations. If NULL and model is a joint_model, the training predictors are used, as the original columns (e.g. a factor stays one column, not its one-hot dummies), which is what the model's predict() takes.
y	A survival object or response variable (e.g. Surv(time, status)). If NULL and model is a joint_model, the original survival object is used.
predict_survival_function	Optional custom prediction function returning an $N \times K$ data frame or matrix of survival probabilities $S(t)$.
predict_risk_function	Optional custom prediction function returning a numeric vector of risk scores or cumulative hazard.
times	A numeric vector of evaluation time points. If NULL, defaults to unique event times in y.
label	Character string naming the model in plots and summaries.
verbose	Logical; if TRUE, progress messages are shown. Default is FALSE.
...	Additional arguments passed to survex::explain_survival().

Value

A survex_explainer object compatible with survex::model_parts(), survex::model_performance(), and survex::predict_parts().

Examples

```
## Not run:
if (requireNamespace("survex", quietly = TRUE) && requireNamespace("survival", quietly = TRUE)) {
  lung <- survival::lung
  df <- na.omit(lung[, c("time", "status", "age", "sex")])
  fit <- joint_model(df, survival::Surv(time, status) ~ age + sex)
  expl <- cbe_explain_survival(fit)
  mp <- survex::model_parts(expl)
}

## End(Not run)
```

---

cbe_factor_reference *Relevel a Factor's Reference Level*

Description

Standalone helper for choosing the reference (baseline) level of a factor before passing it to [cbe_cox_single()] or [cbe_cox_multi()]. Wraps [stats::relevel()], defaulting to the first level (matching the reference-row convention already used by cbe_cox_single()/cbe_cox_multi()), and attaches a "cbe_reference_level" attribute recording the chosen level.

Usage

```
cbe_factor_reference(x, ref_level = NULL)

factor_reference(x, ref_level = NULL)
```

Arguments

x	A factor, or a vector coercible to one via as.factor().
ref_level	Character string naming the level to use as reference. Defaults to the first level of x (its current or natural ordering).

Details

cbe_cox_single() and cbe_cox_multi() are not rewired to require this helper; it is an optional convenience for callers who want to choose the reference level explicitly before fitting.

Value

The releveled factor, with attribute "cbe_reference_level" set to the chosen reference level.

See Also

[cbe_cox_single()], [cbe_cox_multi()]

---

cbe_four_quadrant_report

Standard 4-Quadrant Clinical Contingency Report

Description

Generates a standard 4-quadrant clinical report for a 2x2 contingency table:

```
q1 | q2
q3 | q4
p = pformat
```

where q1 is (Row 1, Col 1), q2 is (Row 1, Col 2), q3 is (Row 2, Col 1), and q4 is (Row 2, Col 2). If any cell count is zero, hypothesis testing automatically defaults to the mid-p version of Central Fisher's exact test via cbe_exact2x2.

Usage

```
cbe_four_quadrant_report(
  data,
  var1 = NULL,
  var2 = NULL,
  label1 = NULL,
  label2 = NULL,
  test = c("auto", "exact", "chisq", "fisher"),
  correct = FALSE,
  p_format = NULL,
  ...
)
```

Arguments

data	A data frame or a 2x2 matrix/table.
var1	Character string of row variable column name (if data is a data frame).
var2	Character string of column variable column name (if data is a data frame).
label1	Display label for row variable.
label2	Display label for column variable.
test	Hypothesis test engine: "auto" (default CBE hierarchy: mid-p exact if zero-cell, otherwise Central Fisher's exact), "exact" (force exact test), "chisq" (force Pearson's Chi-squared test via chisq.test), or "fisher" (force fisher.test).
correct	Logical; whether to apply continuity correction for "chisq" (default FALSE).
p_format	Function for p-value formatting (default pformat).
...	Additional arguments passed to cbe_contingency_plot.

Value

A list containing:

quadrants	A tibble with counts and percentages for q1, q2, q3, and q4.
p_value	Numeric exact test p-value.
p_formatted	Formatted p-value string (e.g. "p = 0.024").
test_method	Name of the exact test used.
compact_report	Compact 3-line string formatted as q1 | q2 // q3 | q4 // p = pformat.
text_card	Console-ready formatted 4-quadrant text card.
plot	A ggplot object rendering the 4-quadrant square report.

Examples

```
tab <- matrix(c(14, 6, 4, 16), nrow = 2,
              dimnames = list(c("Yes", "No"), c("Active", "Placebo")))
rep <- cbe_four_quadrant_report(tab, label1 = "Response", label2 = "Treatment")
cat(rep$compact_report, "\n\n")
cat(rep$text_card)

# Force Pearson Chi-squared test:
rep_chi <- cbe_four_quadrant_report(tab, test = "chisq")
cat(rep_chi$compact_report, "\n")
```

---

cbe_heatmap_plot *Heatmap Plot for 2-Way Contingency Tables*

Description

Dedicated wrapper for cbe_contingency_plot with type = "heatmap".

Usage

```
cbe_heatmap_plot(
  data,
  var1 = NULL,
  var2 = NULL,
  label1 = NULL,
  label2 = NULL,
  fill_colors = c("#F4F4F4", "#9D2235"),
  show_counts = TRUE,
  title = NULL,
  ...
)
```

Arguments

data	A data frame containing the categorical variables, or a 2-way table/matrix.
var1	Character string of the row variable column name (ignored if data is a table).
var2	Character string of the column variable column name (ignored if data is a table).
label1	Optional display label for var1 (defaults to var1).
label2	Optional display label for var2 (defaults to var2).
fill_colors	Character vector of colors for the palette or gradient. Defaults to Temple University brand palette (c("#F4F4F4", "#9D2235")).
show_counts	Logical (default TRUE); whether to print cell counts inside plots.
title	Plot title. If NULL, defaults to "{label1} \u00d7 {label2}".
...	Additional arguments passed to underlying plotting functions.

Value

A ggplot object.

---

cbe_km_single *Univariable Kaplan-Meier and Cox Screening Engine*

Description

Pairs a univariable [cbe_cox_single()] Cox proportional hazards fit with the matching Kaplan-Meier stratified survival curve for the same predictor, so both are produced from a single call without refitting the Cox model twice.

Usage

```
cbe_km_single(data, outcome = "outcome", feature, conf_level = 0.95)

km_single(data, outcome = "outcome", feature, conf_level = 0.95)
```

Arguments

data	A data frame containing survival data and the candidate feature.
outcome	Character string naming the Surv object or outcome column in data (e.g., "outcome" or "Surv(survival_time, status)").
feature	Character string naming the candidate predictor column in data. Factor (or coercible) predictors are used directly as Kaplan-Meier strata; numeric predictors are binned into quartiles for the Kaplan-Meier strata (matching [plot_cox_survival()]'s convention), while the Cox side treats them as continuous.
conf_level	Numeric confidence level (default: 0.95).

Value

An object of class cbe_km containing:

- cox: The cbe_cox object from [cbe_cox_single()].
- km_fit: The survival::survfit object for the Kaplan-Meier curve.
- km_tidy: broom::tidy() table of km_fit.
- direction: "increases" or "decreases", pulled from the Cox hazard ratio direction.
- summary: One-row-per-stratum tidy summary combining Cox hazard ratios and Kaplan-Meier median survival.
- feature: Feature name string.
- var_label: Human-readable variable label.

See Also

[cbe_cox_single()], [plot_cox_survival()]

---

cbe_labels *Get or Set Dataset and Database Labels*

Description

Extends **labelled** variable-level conventions to dataset-level and database-level metadata attributes.

Usage

```
cbe_dataset_label(x)

cbe_dataset_label(x) <- value

cbe_database_name(x)

cbe_database_name(x) <- value

cbe_database_label(x)

cbe_database_label(x) <- value
```

```
cbe_set_dataset_labels(x, labels)

cbe_get_dataset_labels(x)

dataset_label(x)

dataset_label(x) <- value

database_name(x)

database_name(x) <- value

database_label(x)

database_label(x) <- value

set_dataset_labels(x, labels)

get_dataset_labels(x)
```

Arguments

x	A data frame (for dataset functions) or a named list of data frames (for database functions).
value	A character string specifying the label or name.
labels	A named character vector or list mapping dataset names to dataset labels.

Value

For getters, the label or name string (or NULL if unset). For setters, the modified object x with the attribute attached.

Examples

```
df <- mtcars
cbe_dataset_label(df) <- "1974 Motor Trend Car Road Tests"
cbe_dataset_label(df)
```

---

cbe_loco_mp_coxnet *Leave-One-Covariate-Out Inference with MiniPatch Ensembles (LOCO-MP) for Cox Models*

Description

Implements the LOCO-MP statistical inference framework (Gan, Zheng, & Allen 2022) for penalized Cox proportional hazards regression. Combines random minipatch ensembling (subsampling both observations/subjects and covariates) with out-of-bag Leave-One-Covariate-Out evaluation to produce distribution-free feature importance estimates, standard errors, asymptotic z-tests, and confidence intervals for survival models.

Usage

```
cbe_loco_mp_coxnet(
  formula = NULL,
  data = NULL,
  x = NULL,
  y = NULL,
  B = 100,
  n_ratio = 0.7,
  m_ratio = 0.5,
  penalty = NULL,
  mixture = 1,
  eval_time = NULL,
  subject_id = NULL,
  alpha = 0.05,
  p_adjust = c("bonferroni", "BH", "fdr", "holm", "none"),
  trunc = 0.05,
  parallel = FALSE,
  seed = NULL,
  covariates = c("path", "baseline"),
  ...
)

loco_mp_coxnet(
  formula = NULL,
  data = NULL,
  x = NULL,
  y = NULL,
  B = 100,
  n_ratio = 0.7,
  m_ratio = 0.5,
  penalty = NULL,
  mixture = 1,
  eval_time = NULL,
  subject_id = NULL,
  alpha = 0.05,
  p_adjust = c("bonferroni", "BH", "fdr", "holm", "none"),
  trunc = 0.05,
  parallel = FALSE,
  seed = NULL,
  covariates = c("path", "baseline"),
  ...
)
```

Arguments

formula	A formula with a Surv() outcome, e.g. Surv(time, status) ~ x1 + x2, or a [recipes::recipe()] whose outcome is a Surv column. A formula may not use ‘strata()‘ or ‘offset()‘; see [coxnet()].
data	A data frame containing the variables in the model.
x	An optional predictor matrix (used if formula and data are NULL).
y	An optional Surv outcome object (used if formula and data are NULL).

B	Number of minipatches to generate. Default is 100.
n_ratio	Subsampling fraction for subjects/observations without replacement. Default is 0.7.
m_ratio	Subsampling fraction for predictors without replacement. Default is 0.5.
penalty	Penalty parameter for coxnet. If NULL, each minipatch is predicted at the smallest penalty on its own glmnet path, an almost unpenalized fit.
mixture	Elastic-net mixing parameter ($\alpha \in [0, 1]$). Default is 1 (lasso).
eval_time	Numeric vector of evaluation time points for IPCW Brier loss integration. If NULL, defaults to deciles of event times.
subject_id	Optional character column name identifying subjects for clustered / counting-process start/stop data. With a recipe, a single column with role "id" is used when NULL, as in [cv_coxnet()].
alpha	Significance level: the two-sided confidence intervals have level 1 - alpha, and autoplot() calls a predictor significant when its adjusted one-sided p-value is below alpha. Default is 0.05.
p_adjust	Multiple testing correction method for the one-sided p-values. Options include "bonferroni", "BH", "fdr", "holm", or "none". Default is "bonferroni".
trunc	Lower truncation threshold for Kaplan-Meier censoring weights. Default is 0.05.
parallel	Logical; if TRUE and future is available, runs patches in parallel.
seed	Optional random seed for reproducible subsampling. The caller's RNG state is restored on exit.
covariates	"path" (default) or "baseline": how the out-of-bag survival curve of a subject with start/stop rows uses the covariates, as in cv_coxnet. With "path" the cumulative hazard is integrated along the subject's covariate path, so no value from after an evaluation time predicts survival at it; "baseline" uses the first interval's covariates. The two agree for right-censored data.
...	Additional arguments passed to coxnet(), and so to glmnet, such as cox.ties, standardize, or penalty.factor. Not weights or offset.

Details

Tests and intervals. For each predictor, ‘importance‘ is the mean, over the subjects that were out of bag in patches both with and without it, of the excess IPCW integrated Brier loss when it is left out. ‘p_value‘ is from a *one-sided* z test of ‘importance > 0‘, the upper tail of the standard normal at ‘statistic‘, and ‘p_adjusted‘ adjusts those one-sided p-values with ‘p_adjust‘. ‘conf_low‘ and ‘conf_high‘ are a *two-sided* interval at level ‘1 - alpha‘, not adjusted for multiple testing. The p-values and the interval therefore answer different questions: the interval excludes 0 when the unadjusted one-sided p-value is below ‘alpha / 2‘, whereas ‘p_adjusted < alpha‘ is a one-sided, adjusted criterion. (With ‘p_adjust = "none"‘, a predictor whose ‘p_value‘ is between ‘alpha / 2‘ and ‘alpha‘ passes the test but has an interval that still contains 0.) [autoplot()][autoplot.cbe_loco_mp_coxnet] colours a predictor "Significant" by ‘p_adjusted < alpha‘, that is, by the one-sided adjusted test, not by whether its interval excludes 0.

Data. The predictors must be numeric, complete, and finite, and there must be at least 3 of them: each minipatch keeps 2 and leaves at least 1 out. This is checked once before any patch is fit. A minipatch that fails anyway (for example, one whose sampled subjects have no events) is dropped with a warning that gives the count and the first error; ‘B‘ in the result is the number that succeeded.

Value

An S3 object of class "cbe_loco_mp_coxnet" containing:

results	A tibble with columns term, importance, std_error, statistic, p_value (one-sided, for importance > 0), p_adjusted, conf_low, conf_high (a two-sided interval at level 1 - alpha); see Details.
eval_time	Evaluation time grid used for IPCW loss integration.
B	Number of successfully fitted minipatches; a warning says when some failed.
n_ratio, m_ratio	Subsampling ratios used.
alpha	Significance level.
p_adjust	Multiple testing adjustment method.
covariates	How start/stop covariates were used; see covariates.

See Also

[nested_cv_coxnet()], [cv_coxnet()], [coxnet()]

Examples

```
if (requireNamespace("glmnet", quietly = TRUE) && requireNamespace("survival", quietly = TRUE)) {
  set.seed(42)
  n <- 60
  df <- data.frame(
    time = stats::rexp(n, 0.1) + 0.1,
    status = stats::rbinom(n, 1, 0.6),
    x1 = stats::rnorm(n),
    x2 = stats::rnorm(n),
    x3 = stats::rnorm(n)
  )
  fit <- cbe_loco_mp_coxnet(survival::Surv(time, status) ~ x1 + x2 + x3, data = df, B = 20)
  fit
  generics::tidy(fit)
}
```

---

cbe_mosaic_plot *Mosaic Plot for 2-Way Contingency Tables*

Description

Dedicated wrapper for cbe_contingency_plot with type = "mosaic".

Usage

```
cbe_mosaic_plot(
  data,
  var1 = NULL,
  var2 = NULL,
  label1 = NULL,
  label2 = NULL,
```

```
  show_counts = TRUE,
  title = NULL,
  ...
)
```

Arguments

data	A data frame containing the categorical variables, or a 2-way table/matrix.
var1	Character string of the row variable column name (ignored if data is a table).
var2	Character string of the column variable column name (ignored if data is a table).
label1	Optional display label for var1 (defaults to var1).
label2	Optional display label for var2 (defaults to var2).
show_counts	Logical (default TRUE); whether to print cell counts inside plots.
title	Plot title. If NULL, defaults to "{label1} \u00d7 {label2}".
...	Additional arguments passed to underlying plotting functions.

Value

A ggplot object.

---

cbe_pairwise_combos *Generate All Pairwise Categorical Combinations*

Description

Helper for report generation to enumerate all pairwise combinations of categorical variables in a dataset with sequential integer IDs and labels.

Usage

```
cbe_pairwise_combos(data, cols = NULL, labels_df = NULL)

pairwise_combos(data, cols = NULL, labels_df = NULL)
```

Arguments

data	A data frame or tibble.
cols	Optional character vector of column names to include. If NULL, all factor and character columns are included.
labels_df	Optional lookup data frame with columns variable and label.

Value

A tibble with columns id, var1, var2, label1, label2, and comparison_label.

Examples

```
df <- data.frame(
  A = factor(c("X", "Y")),
  B = factor(c("1", "2")),
  C = factor(c("M", "N"))
)
cbe_pairwise_combos(df)
```

`cbe_palette` *Temple CBE Color Palette*

Description

Official and complementary colors for CBE publications and slide decks.

Usage

```
cbe_palette
```

`cbe_predict_parts_shap` *SHAP Variable Attributions across Survival and Joint Models*

Description

Computes individual observation SHAP attributions: using time-dependent `SurvSHAP(t)` via `survex::predict_parts(` `= "survshap")` for survival explainers, and `DALEX::predict_parts(type = "shap")` for classification/regression explainers.

Usage

```
cbe_predict_parts_shap(
  explainer,
  new_observation,
  type = c("auto", "shap", "survshap"),
  ...
)

predict_parts_shap(
  explainer,
  new_observation,
  type = c("auto", "shap", "survshap"),
  ...
)
```

Arguments

explainer	An explainer object produced by cbe_explain_survival(), cbe_explain(), or DALEX::explain().
new_observation	A 1-row data frame containing the observation to explain.
type	Character specifying the attribution type: "auto" (the default), "shap", or "survshap".
...	Additional arguments passed to survex::predict_parts() or DALEX::predict_parts().

Value

A predict_parts attribution object, printable and plottable with plot().

---

cbe_sas_macro_dir *Directory of TempleCBE SAS Macros*

Description

Returns the file system path to the directory containing TempleCBE's bundled SAS macros (‘cbe_brier_score.sas‘, ‘cbe_cox_phreg.sas‘, ‘cbe_counting_process.sas‘, and ‘cbe_macros.sas‘).

Usage

```
cbe_sas_macro_dir()

sas_macro_dir()
```

Value

Absolute path to the SAS macro directory.

See Also

[cbe_sas_macro_path()], [run_sas_script()], [find_sas()]

Examples

```
try(cbe_sas_macro_dir())
```

cbe_sas_macro_path *Path to a TempleCBE SAS Macro File*

Description

Returns the absolute path to an installed TempleCBE SAS macro or benchmark script.

Usage

```
cbe_sas_macro_path(macro = "cbe_macros.sas")

sas_macro_path(macro = "cbe_macros.sas")
```

Arguments

macro	Filename of the SAS macro or script (default: ‘"cbe_macros.sas"‘). Available files include ‘"cbe_macros.sas"‘, ‘"cbe_brier_score.sas"‘, ‘"cbe_cox_phreg.sas"‘, ‘"cbe_counting_process.sas"‘, ‘"coxtvc.sas"‘, ‘"cpdata.sas"‘, ‘"benchmark_brier_lung.sas"‘, ‘"benchmark_brier_lung_full.sas"‘, and ‘"example_85_7.sas"‘.

Value

Absolute path to the requested ‘.sas‘ file.

See Also

[cbe_sas_macro_dir()], [run_sas_script()], [find_sas()]

Examples

```
try(cbe_sas_macro_path("cbe_macros.sas"))
```

cbe_square_plot *Square Plot: Standard 4-Quadrant Contingency Report*

Description

Dedicated wrapper for cbe_contingency_plot with type = "square". For 2x2 contingency tables, renders the standard 4-quadrant report:

```
q1 | q2
q3 | q4
p = pformat
```

If any cell count is zero, hypothesis testing automatically defaults to the mid-p version of Central Fisher's exact test via cbe_exact2x2.

Usage

```
cbe_square_plot(
  data,
  var1 = NULL,
  var2 = NULL,
  label1 = NULL,
  label2 = NULL,
  test = c("auto", "exact", "chisq", "fisher"),
  correct = FALSE,
  engine = c("ggplot", "corrplot"),
  size_range = c(6, 22),
  fill_colors = c("#F4F4F4", "#9D2235"),
  show_counts = TRUE,
  title = NULL,
  caption = NULL,
  p_format = NULL,
  ...
)
```

Arguments

data	A data frame containing the categorical variables, or a 2-way table/matrix.
var1	Character string of the row variable column name (ignored if data is a table).
var2	Character string of the column variable column name (ignored if data is a table).
label1	Optional display label for var1 (defaults to var1).
label2	Optional display label for var2 (defaults to var2).
test	Hypothesis test engine: "auto" (default CBE hierarchy), "exact" (force Central Fisher exact test for 2x2 or simulated Fisher for RxC), "chisq" (force Pearson's Chi-squared test via chisq.test), or "fisher" (force fisher.test).
correct	Logical; whether to apply continuity correction when test = "chisq" (default FALSE following CBE standard protocol).
engine	Engine for balloon/square rendering: "ggplot" (default), "corrplot", or "ggpubr".
size_range	Numeric vector of length 2 giving min and max glyph sizes for balloon and square plots (default c(6, 22)).
fill_colors	Character vector of colors for the palette or gradient. Defaults to Temple University brand palette (c("#F4F4F4", "#9D2235")).
show_counts	Logical (default TRUE); whether to print cell counts inside plots.
title	Plot title. If NULL, defaults to "{label1} \u00d7 {label2}".
caption	Optional plot caption. If NULL, automatically computed and formatted using pformat with test description.
p_format	Optional p-value formatting function (default pformat).
...	Additional arguments passed to underlying plotting functions.

Value

A ggplot object or invisibly the contingency table.

cbe_survex_loss_ibs *TempleCBE Custom Loss Functions for survex*

Description

Provides survival loss functions implementing TempleCBE's inverse-probability-of-censoring weighted (IPCW) Graf Integrated Brier Score and time-dependent Brier Score for use with `survex::model_parts()` and `survex::model_performance()`.

Usage

```
cbe_survex_loss_ibs(
  normalization = c("none", "t_max", "survival"),
  max_quantile = 1,
  trunc = 0.05
)

cbe_survex_loss_brier(trunc = 0.05)

survex_loss_ibs(
  normalization = c("none", "t_max", "survival"),
  max_quantile = 1,
  trunc = 0.05
)

survex_loss_brier(trunc = 0.05)
```

Arguments

`normalization`	Optional normalization parameter for `survex` integral calculation. Can be `NULL`, `"t_max"`, or `"survival"`.
`max_quantile`	Upper quantile cutoff for evaluation time window. Default is 1.
`trunc`	Lower truncation threshold for censoring survival probabilities to prevent inflation by rare tails. Default is 0.05.

Value

A loss function with attributes `"loss_type"` (`"integrated"` or `"time-dependent"`) and `"loss_name"`, directly passable to `survex`.

Examples

```
## Not run:
if (requireNamespace("survex", quietly = TRUE)) {
  loss_ibs <- cbe_survex_loss_ibs()
  # mp <- survex::model_parts(explainer, loss_function = loss_ibs)
}

## End(Not run)
```

`cbe_test_categorical` *Institutional Categorical Hypothesis Test for gtsummary and Batch Testing*

Description

Standard categorical hypothesis testing engine conforming to the CBE statistical protocol. Operates as a drop-in custom test for `add_p` or as a standalone batch testing engine with optional **furrr** parallelization.

Usage

```
cbe_test_categorical(
  data,
  variable = NULL,
  by = NULL,
  test = c("auto", "exact", "chisq", "fisher"),
  correct = FALSE,
  B = 2000L,
  simulate.p.value = NULL,
  parallel = FALSE,
  n_chunks = 4L,
  group = NULL,
  type = NULL,
  test.args = NULL,
  adj.vars = NULL,
  conf.level = NULL,
  tbl = NULL,
  continuous_variable = NULL,
  ...
)

test_categorical(
  data,
  variable = NULL,
  by = NULL,
  test = c("auto", "exact", "chisq", "fisher"),
  correct = FALSE,
  B = 2000L,
  simulate.p.value = NULL,
  parallel = FALSE,
  n_chunks = 4L,
  group = NULL,
  type = NULL,
  test.args = NULL,
  adj.vars = NULL,
  conf.level = NULL,
  tbl = NULL,
  continuous_variable = NULL,
  ...
)
```

Arguments

data	Data frame supplied directly or by **gtsummary**.
variable	Character string column name for the feature being tested, a character vector of variable names for batch testing, or a two-sided formula of the form response ~ grouping_var or var1 + var2 ~ grouping_var. If NULL, all categorical/factor columns in data (excluding by) are tested.
by	Character string column name for the stratifying/grouping variable. Can be omitted when variable is a formula.
test	Hypothesis test engine: "auto" (default CBE hierarchy), "exact" (force Central Fisher exact test for 2x2 or simulated Fisher for RxC), "chisq" (force Pearson's Chi-squared test via chisq.test), or "fisher" (force fisher.test).
correct	Logical; whether to apply continuity correction when test = "chisq" (default FALSE following CBE standard protocol).
B	Integer; number of Monte Carlo replicates when Fisher's exact test with simulation is triggered (default 2000L).
simulate.p.value	Optional logical; if specified, explicitly enables or disables Monte Carlo simulation for Fisher's test. If NULL (default), auto-triggers simulation when table total exceeds 500 or any dimension exceeds 2.
parallel	Logical; whether to use **furrr** for parallel execution (for batch testing across multiple variables or chunking high-B simulations). Requires setting a plan beforehand.
n_chunks	Integer; number of chunks to partition B into when running parallel simulations with B >= 10000 (default 4L).
group, type, test.args, adj.vars, conf.level, tbl, continuous_variable	Unused; accepted and ignored so this function satisfies **gtsummary**'s custom-test calling convention (see tests) without those arguments leaking into ... and reaching chisq.test/ fisher.test, neither of which accepts extra named arguments.
...	Additional arguments passed to exact2x2 or fisher.test.

Details

The function follows a rigorous 4-rule hierarchy:

1. **2x2 table with zero cell**: Uses Central Fisher's exact test with mid-p adjustment (exact2x2 with midp = TRUE) to avoid extreme conservatism.
2. **2x2 table without zero cell**: Uses Central Fisher's exact test (exact2x2 with midp = FALSE).
3. **RxC table with sparse counts**: If any expected cell count is less than 5, runs Fisher's exact test with Monte Carlo simulation (fisher.test with simulate.p.value = TRUE).
4. **RxC table with adequate counts**: If all expected cell counts are at least 5, runs Pearson's Chi-squared test without continuity correction (chisq.test with correct = FALSE).

Value

When variable is a single string (such as when called by **gtsummary**), returns a one-row tibble compliant with **gtsummary**'s custom test specification (p.value, statistic, parameter, method). When variable is a vector or NULL, returns a tidy tibble of test results across all evaluated variables.

Examples

```
library(gtsummary)
trial |>
  tbl_summary(by = trt, include = c(response, death, grade)) |>
  add_p(test = all_categorical() ~ cbe_test_categorical) |>
  separate_p_footnotes()

# Standalone batch testing across multiple features:
cbe_test_categorical(trial, variable = c("response", "death", "grade"), by = "trt")

# Formula interface:
cbe_test_categorical(trial, response ~ trt)
```

---

cbe_theme_survival *CBE ggplot2 Theme for Survival Analysis Visualizations*

Description

A variant of `theme_cbe()` tuned for Cox model and Kaplan-Meier plots (forest plots, predicted/observed survival curves, marginal risk curves), giving a single consistent look across `plot_cox_forest()`, `plot_cox_forest_multi()`, `plot_cox_survival()`, and `plot_cox_marginal()`.

Usage

```
cbe_theme_survival(base_size = 12, base_family = "")
```

Arguments

`base_size`	Base font size (default: 12)
`base_family`	Base font family (default: "")

Value

A ggplot2 theme object

See Also

[theme_cbe()], [theme_cbe_deck()]

---

cbe_variable_roles *Define and Manage Clinical Variable Roles*

Description

Assigns functional roles (e.g. predictor, outcome, identifier, time variable, strata, weights) to variables across a single data frame or an entire R database list. Produces standardized schemas compatible with `validate_column_mapping`, **recipes**, and modeling workflows.

Usage

```
cbe_variable_roles(
  data,
  id = NULL,
  outcome = NULL,
  time = NULL,
  predictors = NULL,
  strata = NULL,
  weight = NULL,
  ignore = NULL,
  roles = NULL
)

cbe_set_roles(data, ...)

cbe_get_roles(data)

cbe_get_predictors(roles)

cbe_get_outcomes(roles)

cbe_get_id_cols(roles)

cbe_get_time_cols(roles)

variable_roles(
  data,
  id = NULL,
  outcome = NULL,
  time = NULL,
  predictors = NULL,
  strata = NULL,
  weight = NULL,
  ignore = NULL,
  roles = NULL
)

set_roles(data, ...)

get_roles(data)

get_predictors(roles)

get_outcomes(roles)

get_id_cols(roles)

get_time_cols(roles)
```

Arguments

data	A data frame or a named list of data frames (an R database).

id	Character vector of column names acting as subject/patient/cluster identifiers.
outcome	Character vector of column names acting as response/outcomes (Y_var).
time	Character vector of column names acting as longitudinal visit/time variables (Time_var).
predictors	Character vector of column names acting as features/predictors (X_var).
strata	Character vector of column names used for stratification or subgroup analysis.
weight	Character vector of column names used for case weights or survey weights.
ignore	Character vector of column names to ignore/exclude from modeling.
roles	Role table (e.g. from cbe_variable_roles) or data object.
...	Additional arguments passed to cbe_variable_roles.

Value

A tibble with columns columns, role, X_var, Y_var, ID_var, Time_var, and dataset_name (when multi-table).

Examples

```
df <- data.frame(id = 1:5, time = 1:5, sbp = c(120, 130, 115, 140, 125), death = c(0, 0, 1, 0, 1))
roles <- cbe_variable_roles(df, id = "id", time = "time", outcome = "death", predictors = "sbp")
```

---

censoring_km *Kaplan-Meier Estimate of the Censoring Distribution*

Description

Fits the "reverse" Kaplan-Meier estimator $G(t) = P(C > t)$, treating censoring as the event, for inverse-probability-of-censoring weights. Estimate it on the data a model was trained on, and apply it to the data being scored.

Usage

```
censoring_km(truth)

## S3 method for class 'censoring_km'
predict(object, time, left = FALSE, trunc = 0, ...)
```

Arguments

truth	A right-censored [survival::Surv()] object with one element per subject, such as the '.truth' column of [surv_subject_truth()].
object	A 'censoring_km' object.
time	Numeric vector of times.
left	If 'TRUE', the left limit $G(t^-)$, i.e. censoring strictly before 'time'.
trunc	Lower bound applied to G, so weights stay finite.
...	Not used.

Value

An object of class ‘censoring_km‘, with ‘time‘ and ‘surv‘; use ‘predict(object, time)‘ for $G(t)$.

See Also

[graf_weights()]

Examples

```
if (requireNamespace("survival", quietly = TRUE)) {
  cens <- censoring_km(survival::Surv(c(2, 4, 6, 8), c(0, 1, 0, 1)))
  predict(cens, c(1, 2, 5, 7))
}
```

---

check_docx_toolchain *Report Which PDF -> DOCX Backends Are Usable on This Machine*

Description

Safe to call at any time; performs discovery only and converts nothing.

Usage

```
check_docx_toolchain(quiet = FALSE)
```

Arguments

quiet	Suppress the printed diagnosis.

Value

Invisibly, a list with elements `python_with_pdf2docx`, `python_any`, `soffice`, `word_com` and `backend`.

See Also

[convert_pdf_to_docx()], [convert_pdfs_to_docx()]

Examples

```
## Not run:
check_docx_toolchain()

## End(Not run)
```

clean_names *Clean and Standardize Variable Names*

Description

Converts variable/column names to lower snake_case and removes invalid special characters.

Usage

```
clean_names(data)
```

Arguments

data	A data frame, tibble, or character vector of names.

Value

Cleaned data frame or character vector.

Examples

```
clean_names(c("First Name", "ZIP Code", "Total ($)"))
```

clean_publish *Publish a Clean Snapshot of the Current Commit to a Remote*

Description

Publishes the tree of the current commit as a clean commit that carries none of the history behind it, and keeps the full history on `private_branch`. There are two modes. `mode = "standalone"` (the default) makes one commit without a parent and, with `push = TRUE`, force-pushes it over the publish branch of `remote`; it is the first publication, into a new empty repository. `mode = "snapshot"` makes one commit whose parent is the tip of the publish branch of `remote`, which must consist only of commits this function made, and pushes it as an ordinary fast-forward; it is every later release. With `push = FALSE` only the local branches are rewritten. A pre-push hook keeps `private_branch` and every commit that only it contains from being pushed by accident.

Usage

```
clean_publish(
  repo_root = NULL,
  publish_branch = NULL,
  private_branch = "private-history",
  remote = NULL,
  commit_msg = NULL,
  push = TRUE,
  mode = c("standalone", "snapshot"),
  allow_ignored = character(),
  confirm_guard = TRUE,
  allow_other_refs = FALSE,
```

```
  allow_overwrite_history = FALSE,
  allow_submodules = FALSE,
  disarm_push_url = TRUE
)
```

Arguments

repo_root	Path to the git repository root (or any directory of its working tree; it may be a linked worktree and may contain spaces). Defaults to the git toplevel of the working directory (an error when that is not in a repository). The repository that is acted on, its git directory, the current branch and the remotes with their URLs are printed first.
publish_branch	Branch to overwrite with the clean commit. Defaults to master, else main, else the current branch (unless that is private_branch). master/main are checked before the current branch on purpose: running this from an arbitrary feature branch must not silently overwrite that branch with a one-commit history. private_branch itself is refused, since publishing onto it would squash away the full history it exists to keep, and so is a name that differs from it only in case.
private_branch	Local branch that retains full history. Default "private-history". Created from the current commit if it does not exist, otherwise moved forward to the current commit (never backward or sideways, see above).
remote	Name of a configured remote to publish to. Required for push = TRUE and for mode = "snapshot", and deliberately without a default: in a clone of the private repository "origin" is the private repository, and a standalone publish force-pushes one commit over its branch. Its push URL is printed before the push. In snapshot mode the remote must be the one that holds the public snapshots: a remote branch whose history is not made only of clean commits made by this function is refused, because the snapshot would carry that history.
commit_msg	Commit message for the clean commit. Default: "Initial clean commit", or "Snapshot <date>" for mode = "snapshot". The trailer line Clean-Publish: v1 is appended after a blank line.
push	Logical (default TRUE); if FALSE, rewrite the local branches but skip the force-push, as a dry run of the destructive step. (The command-line script scripts/clean_publish.R is the other way round: it pushes only when given --push.)
mode	Mode of publishing: "standalone" (default) creates a root (parentless) commit and force-pushes with lease to the remote; "snapshot" fetches the remote publish branch and creates a fast-forward commit parented on the remote tip.
allow_ignored	Character vector of file paths that are tracked in git and match .gitignore patterns, explicitly allowed to be included in the clean commit. By default, clean_publish stops if any tracked-but-ignored files exist to prevent accidental leakage.
confirm_guard	Logical (default TRUE). A push guard that could not be confirmed by the test pushes is an error before anything moves. FALSE turns that case into a warning; a guard that is shown not to work is an error either way.
allow_other_refs	Logical (default FALSE). Standalone mode with push = TRUE refuses a remote that has refs other than the publish branch (branches, tags, notes, refs/pull/*), because they keep the old history. TRUE goes on and lists them in the success message.

`allow_overwrite_history`
: Logical (default `FALSE`). Standalone mode with `push = TRUE` refuses a remote that holds this repository's own history. `TRUE` overwrites the branch all the same.

`allow_submodules`
: Logical (default `FALSE`). A current commit with submodules (gitlinks or a tracked '`.gitmodules`') is refused, because their URLs and commit ids would be published. `TRUE` publishes them and prints their URLs.

`disarm_push_url`
: Logical (default `TRUE`). After a successful push, record the real push URL of `remote` and replace its `remote.<remote>.pushurl` by an unusable value (see above). `FALSE` leaves the remote as it was; the pre-push hook is then the only guard of pushes by the name of the remote.

Details

The target must be a NEW EMPTY repository. A force-push replaces one branch of the remote and nothing else: its other branches, its tags, its notes and GitHub's hidden `refs/pull/*` keep the commits they point to, so a repository that ever held the private history keeps it readable, and no push, force-push included, can remove a hidden ref. Create the public repository for this purpose, add it to a throwaway clone as the only remote and publish into it (the steps are in '`REVIEW.md`', "Publishing to the public repository").

`remote` is the name of a remote you have added, for example `git remote add public <url>`, and has no default: in a clone of the private repository `origin` is the private repository, and a standalone publish force-pushes one commit over its branch. To publish the same snapshot to a second host, add a second remote (`git remote add med-cbe <url>`) and run again with `remote = "med-cbe"`.

Nothing is changed until every check has passed. The function stops, and leaves branches, the working tree, the git configuration, the hooks and the remote as they were, when

- git has no identity for the clean commit (`git var GIT_AUTHOR_IDENT` or `GIT_COMMITTER_IDENT` fails, as in a fresh clone);
- a branch name is not valid as `refs/heads/<name>`, starts with an at sign or contains an at sign followed by a brace, or the private and the publish branch differ only in case;
- the working tree has uncommitted changes or untracked files (files ignored by `.gitignore` do not count and are never published: the snapshot is exactly the tree of the current commit, tracked files that are also ignored included);
- the current commit contains submodules (a gitlink or a tracked '`.gitmodules`'), unless `allow_submodules = TRUE`;
- `private_branch` exists but is not contained in the current commit, so that moving it to the current commit would drop commits from every branch. Run the function from `private_branch` (or a branch that contains it), not from the publish branch or an older branch;
- the publish branch, or `private_branch` when it is not the current branch, is checked out in another git worktree;
- `push = TRUE` and `remote` is not a configured remote, has more than one push URL, or its refs cannot be listed, or (standalone mode) holds this repository's history or has refs other than the publish branch (see the section on the remote);
- `mode = "snapshot"` and the history of the remote branch is not made only of clean commits, or the repository is shallow;
- an existing `pre-push` hook is not a shell script, or the hook directory lies inside the working tree and is not ignored.

Every git call a check depends on stops the run when it fails; none is read as "nothing found". After the checks the hook and the configuration are written and the guard is tested (see below); if that fails, branches and remote are still untouched.

The identity that is published. The clean commit takes its author and committer, and the time zone offset of its dates, from git's own settings (`user.name`, `user.email`, the `GIT_AUTHOR_*` and `GIT_COMMITTER_*` environment variables, dates included). They are printed before anything changes; set them in the environment of the R session to publish another identity.

The remote. `push = TRUE` looks at the remote before anything changes.

- The real push URL is read with `git remote get-url --push --all`; a remote with more than one push URL is refused. The push goes to that URL (`git push <url> <commit>:refs/heads/<branch>`), not to the name of the remote, and every URL that is printed or reported has its credentials removed.
- Standalone mode lists *all* refs of the remote (`git ls-remote`; a failing listing is an error) and refuses, unless `allow_other_refs = TRUE`, when there is any ref besides the publish branch. A force-push replaces one branch only: tags, other branches, notes and GitHub's hidden `refs/pull/*` keep the old commits, and the last cannot be removed at all. **Publish into a new, empty repository.** It also refuses, unless `allow_overwrite_history = TRUE`, when the tip of a branch of the remote is a commit that this repository has (reachable from the commit being published, from the private or the publish branch or from a recorded private tip) and that is not a clean commit: the remote then holds this repository's history, as `origin` does in a clone of the private repository.
- The standalone push is a force-push with an explicit lease on what the listing showed: `--force-with-lease=refs/` (empty when the branch does not exist), so it fails when the branch moved in the meantime and does not depend on a possibly stale remote-tracking ref.
- Snapshot mode builds on the tip of the remote branch, so that history is part of the snapshot. Every commit this function makes ends with the trailer line `Clean-Publish: v1`, and snapshot mode refuses a remote branch unless its whole history is linear, has one parentless root and consists of such commits (otherwise the message gives the number of commits that lack the trailer and the newest of them). A remote that holds the private repository, a rewritten archive of it or someone else's commits is refused; a shallow repository is refused too, because it cannot show the history.
- After a successful push the real push URL is recorded in the git config key `cleanpublish.<remote>.realpushurl` and `remote.<remote>.pushurl` is set to `DISABLED-use-clean_publish`, which no client can use (unless `disarm_push_url = FALSE`). A later `git push <remote>`, `git push --no-verify`, a different `core.hooksPath` and **gert** (which **usethis** uses) then fail instead of publishing private history; fetching is not affected, and later runs of this function read the recorded URL. To undo it: `git remote set-url --push <remote> <url>`. An explicit push to the real URL, `git send-pack` and other tools that do not read the remote's configuration still work.
- The success message states only what is guaranteed: the publish branch of the remote now points to the clean commit. It lists the other refs of the remote, which keep whatever they point to. If a step fails after the push, the message says that the remote is already updated and names the clean commit.

Safety net. Before a branch is moved, the old tips of the private and the publish branch are kept as `refs/backup/clean_publish/<UTC time>/<branch>`. To restore one, run `git branch -f <branch> <that ref>` from another branch; git refuses it for the branch that is checked out, and the private branch is the checked-out one after every run, so there run `git checkout <other branch>` first, or `git reset --hard <that ref>`. The remote is updated first and the local branches are only moved afterwards, so a failed push leaves the local branches untouched; if any later step fails, the branch that was checked out is checked out again. After the moves the function checks

that both branches are where it put them (on a case-insensitive file system a name that differs in case is the same branch) and stops with the backup refs when they are not.

The push guard. The hook is installed where git reads it (the `hooks/pre-push` of `git rev-parse --git-path`: shared by linked worktrees, and honouring `core.hooksPath`). It goes into an existing hook for `sh`, `dash`, `bash` or `ksh` as a clearly marked block right after its first line, so the hook keeps working (it still receives its input, and none when nothing is pushed), and that block is re-written, not duplicated, on the next run. A hook in another language, `zsh` included, stops the run with a message, and the hook an earlier version of this function wrote is replaced; no other hook is replaced or skipped silently. The private history is protected by commit id, whatever becomes of the branch names. Before it pushes or moves anything, the function records every commit that holds private history – the commit it publishes and the earlier tips of the private and the publish branch – as a full id in the multi-valued git config key `cleanpublish.privatetip` (never removed), and the names of the private branches in `cleanpublish.protectedbranch` (one per `private_branch` ever used). The hook refuses to push

- any ref named like a protected branch;
- any ref, tag or other branch from which a commit can be reached that the live tip of a protected branch or a recorded private tip reaches, and that is not part of a clean commit made by this function (their ids are the values of `cleanpublish.cleancommit`). Pushing a tag on a private commit, an object id, the publish branch after merging the private branch into it, or `--mirror` is therefore refused too, also after the private branch was renamed, deleted or moved, and after a first publish whose push failed;
- anything that is not made of clean commits, when protected branches are configured and neither they nor a recorded tip can be found in the repository (the message says how to release the guard: `git config --unset-all cleanpublish.protectedbranch`);
- a tag, tree or blob that does not lead to a commit, unless a clean commit has it.

Both keys take full commit ids and nothing else: a revision such as a branch name or an abbreviated id is ignored. See `guard_public_remote()` for the fence on the public URL and for `TEMPLECBE_PUBLISHING=1`, which this function sets for its own push and which lets through nothing but the clean commits it recorded.

The guard is confirmed before anything moves. After installing the hook the function pushes to a temporary local repository: the private branch name, and the commit it publishes under another branch name and as a lightweight tag. Each push must be refused by the hook. It stops with an error, before any branch moves or anything is pushed, when a push is accepted (the hook does not run, or knows only the branch name), and also when a push is refused for any other reason, because then the guard is unconfirmed. Use `confirm_guard = FALSE` to go on with a warning when a push was refused for another reason and that cannot be helped.

What the guard does not stop. It is a hook of command-line `git`, so it does not stop `git push --no-verify`, a push made by a client that does not run hooks (the **gert** package, which **usethis** uses, is one), a push after `core.hooksPath` has been pointed elsewhere or the hook has been deleted, `git send-pack`, or private commits that no recorded tip and no protected branch reaches (history that was created after the last run and is not an ancestor of a recorded tip). Disarming the push URL (see above) stops the first two and the last client for pushes by the name of the remote, not for a push to its real URL. The guard also needs the objects: if `git gc` has removed the commit of a recorded tip, that tip is ignored with a note. The safest procedure is to publish from a throwaway clone and to keep no pushable public remote in the clone you work in.

The command line. `Rscript scripts/clean_publish.R --help` lists the options of the same function. Unlike `clean_publish()` it is a dry run unless `--push` is given (`--snapshot` selects the snapshot mode, `--remote NAME` names the remote and is required for both, `--fence NAME|URL` calls `guard_public_remote()` and takes no other option). An unknown or mistyped option is an error.

It runs with `options(warn = 1)`, and a warning raised before the push stops the run: nothing is pushed and the exit status is 1 (2 for a bad command line, 0 for success). The logic is the internal function `clean_publish_cli()`; the script only loads the package and calls it.

Value

The clean commit SHA, invisibly.

Examples

```
## Not run:
clean_publish(push = FALSE) # rewrite locally, send nothing
clean_publish(remote = "public")
clean_publish(remote = "med-cbe")
clean_publish(remote = "public", mode = "snapshot")

## End(Not run)
```

---

collapse_by_stage *Collapse an igraph pipeline graph to one node per pipeline 'stage'*

Description

The full per-file compute graph (~90+ nodes) is the right level of detail for the interactive HTML, where a reader can zoom and pan – but it cannot be made legible as a single static image for the PDF/DOCX deliverable at any reasonable page size. This produces a bird's-eye version instead: one node per 'stage', with an edge between two stages whenever any file in the first stage feeds any file in the second (parallel edges collapsed).

Usage

```
collapse_by_stage(g, stage_labels = NULL)
```

Arguments

g	An igraph object (see as_igraph()).
stage_labels	Optional named character vector mapping stage identifiers to display labels.

Value

An igraph object with one node per stage.

`compute_graph` *Computational Pipeline Dependency Graphs*

Description

S4 classes and helper functions for tracking file-level dependencies in an analysis pipeline (raw data, rendering scripts, derived artifacts) and for visualizing, summarizing, and topologically sorting the resulting dependency graph. Language-, format-, and project-agnostic: a node is any file on disk (R, Python, SAS, Quarto, spreadsheets, ...), identified by path rather than tied to a specific tool.

Details

The graph engine (**igraph/tidygraph**) is a hard dependency. Three rendering backends are optional (Suggests) and only required by the functions that use them: static PNG export (`print_pipeline`) needs **ggraph**; the interactive Graphviz view (`visualize_pipeline`) needs **DiagrammeR**; and the interactive HTML export (`visualize_pipeline_interactive`, `export_interactive_pipeline`, `export_subgraph`) needs **visNetwork** and **htmlwidgets**.

`controls_datasets` *Check if a FileOutputs object controls datasets (outputs an rds, xlsx, etc. file)*

Description

Check if a FileOutputs object controls datasets (outputs an rds, xlsx, etc. file)

Usage

```
controls_datasets(obj)
```

Arguments

`obj`	A FileOutputs object.

Value

Logical.

`control_race_survival` *Control Parameters for Racing Survival Workflows*

Description

Creates a racing control specification via `control_race` with default settings optimized for clinical survival workflows and nested cross-validation.

Usage

```
control_race_survival(
  save_pred = TRUE,
  parallel_over = c("everything", "resamples", "across"),
  save_workflow = TRUE,
  burn_in = 3,
  num_ties = 10,
  alpha = 0.05,
  randomize = TRUE,
  ...
)

cbe_control_race(
  save_pred = TRUE,
  parallel_over = c("everything", "resamples", "across"),
  save_workflow = TRUE,
  burn_in = 3,
  num_ties = 10,
  alpha = 0.05,
  randomize = TRUE,
  ...
)
```

Arguments

`save_pred`	Logical; whether to save out-of-fold assessment predictions (default `TRUE`).
`parallel_over`	How to parallelize execution: `"everything"` (default), `"resamples"`, or `"across"`.
`save_workflow`	Logical; whether to retain the fitted workflow object in the output (default `TRUE`), required by `fit_best`.
`burn_in`	Minimum number of resamples evaluated before candidate configurations can be eliminated (default 3).
`num_ties`	Number of bootstrap samples used to evaluate ties in win-fraction racing (default 10).
`alpha`	Significance level threshold for elimination in ANOVA racing (default 0.05).
`randomize`	Logical; whether to randomize the order of resamples (default `TRUE`).
`...`	Additional arguments forwarded to `control_race`.

Value

A `control_race` object.

See Also

tune_race_survival, control_race

Examples

```
## Not run:
if (requireNamespace("finetune", quietly = TRUE)) {
  ctrl <- control_race_survival()
}

## End(Not run)
```

---

convert_pdfs_to_docx *Convert PDFs to DOCX Using the Best Available Backend*

Description

Backend preference is python (pdf2docx) -> LibreOffice -> Word COM (Windows only). When the python backend fails on an individual file and LibreOffice is present, that file is retried once with LibreOffice.

Usage

```
convert_pdfs_to_docx(
  conversions,
  backend = "auto",
  strict = TRUE,
  timeout = 600
)
```

Arguments

conversions	A data frame with character columns src and dest, and optionally temp_dest, a second location each converted file is copied to.
backend	One of "auto", "python", "libreoffice", "word_com".
strict	If TRUE (default), stop at the first bad row and, when every row is fine, error with an actionable message if no backend is available. Set FALSE to warn and skip instead; a skipped row comes back with converted = FALSE.
timeout	Seconds a converter may take on one file before it is stopped; it applies to every backend (python, LibreOffice and Word COM). A file that times out gets a warning that says so and comes back with converted = FALSE (the python backend then retries it once with LibreOffice, which gets its own timeout). Stopping a converter stops the process that was started; any process that one started in turn (LibreOffice's soffice.bin, or the Word instance behind Word COM) may keep running and has to be closed by hand.

Details

The rows of `conversions` are checked before any backend is looked for, so a bad row is reported the same way on every machine, whether or not a converter is installed. A row is bad when its `src` does not exist or is empty (0 bytes), when its `dest` is missing, or when its `dest` or `temp_dest` ends in `.pdf` (a DOCX is not a PDF, and the file that is there would be overwritten). A `src` that resolves to the same file as its `dest` or `temp_dest`, however the path is spelled (`./a.pdf`, another case), and a `dest` that is the same file as its `temp_dest`, are always errors, whatever `strict` is, because converting would overwrite the PDF or empty the copy. Rows that pass are then converted; if none pass, no backend is looked for at all.

A converter writes into a fresh file next to `dest`, and that file replaces `dest` only when the conversion succeeded and left a non-empty file. For pdf2docx and Word COM, succeeded means that the converter exited with status 0. LibreOffice's exit status is not looked at: it writes into a private, empty folder, so whatever it leaves there was written by this run, and a first start with a fresh profile may exit non-zero after a good conversion. A time-out is a failure for every backend. A DOCX that was already at `dest` (from an earlier run, say) is never mistaken for the result: when the conversion fails, `dest` is left exactly as it was (an old file is neither deleted nor reported as converted, and a half-written file from a crashed converter is discarded), and the row comes back with `converted = FALSE`.

Value

`conversions` with an added logical `converted` column.

See Also

[convert_pdf_to_docx()], [check_docx_toolchain()], and [zip_reports()], whose `docx_from_pdf` argument accepts `convert_pdf_to_docx`.

`convert_pdf_to_docx` *Convert a Single PDF to DOCX Using the Best Available Backend*

Description

Convenience wrapper around [convert_pdfs_to_docx()] for one file. Its `src, dest` signature fits the `docx_from_pdf` callback of [zip_reports()].

Usage

```
convert_pdf_to_docx(
  src,
  dest = sub("\\.pdf$", ".docx", src, ignore.case = TRUE),
  backend = "auto",
  strict = TRUE,
  timeout = 600
)
```

Arguments

src	Path to the input PDF file.
dest	Path for the output DOCX file. Defaults to src with .pdf replaced by .docx.
backend	One of "auto", "python", "libreoffice", "word_com".
strict	If TRUE (default), stop at the first bad row and, when every row is fine, error with an actionable message if no backend is available. Set FALSE to warn and skip instead; a skipped row comes back with converted = FALSE.
timeout	Seconds a converter may take on one file before it is stopped; it applies to every backend (python, LibreOffice and Word COM). A file that times out gets a warning that says so and comes back with converted = FALSE (the python backend then retries it once with LibreOffice, which gets its own timeout). Stopping a converter stops the process that was started; any process that one started in turn (LibreOffice's soffice.bin, or the Word instance behind Word COM) may keep running and has to be closed by hand.

Value

Invisibly, dest on success, or FALSE if conversion failed.

Examples

```
## Not run:
convert_pdf_to_docx("report.pdf")
zip_reports(reports, output_formats = c("pdf", "docx"), docx_from_pdf = convert_pdf_to_docx)

## End(Not run)
```

---

copy_data_manifest *Copy Datasets to Snapshot Directory*

Description

Copies files listed in a manifest from their source paths to a destination directory, preserving timestamps and validating contents immediately upon copy. Manifests must come from a trusted source. Source and destination paths are confined to their respective directories and must not contain parent-directory traversals.

Usage

```
copy_data_manifest(
  dir = ".",
  manifest = NULL,
  project_root = here::here(),
  overwrite = FALSE
)
```

Arguments

dir	Destination directory for frozen copies.
manifest	Data frame with columns file and source. If NULL, read from file.path(dir, "manifest.csv").
project_root	Absolute or relative root directory for resolving source paths (default: here::here()).
overwrite	Logical; whether to overwrite existing destination files (default: FALSE).

Value

A validation tibble produced by validate_data_manifest.

---

correlation_diff *Difference in Correlation Matrices Between Two Datasets*

Description

Compares the correlation matrix of comparison_data against that of baseline_data (e.g. a later timepoint vs. baseline, or a treatment group vs. a reference group), matching numeric variables by column name.

Usage

```
correlation_diff(
  baseline_data,
  comparison_data,
  cor.use = "everything",
  cor.method = "pearson",
  na_omit = TRUE
)
```

Arguments

baseline_data	A data frame or tibble treated as the reference.
comparison_data	A data frame or tibble to compare against baseline_data.
cor.use, cor.method, na_omit	See correlation_plot.

Details

Unlike PCA loadings (see pca_loading_diff), correlation coefficients have no sign-ambiguity to correct for – a correlation matrix is uniquely determined by the data, so the difference is simply (comparison correlation) minus (baseline correlation), with no sign-alignment step needed.

Variables are matched by name (numeric column names shared by both datasets). If the two datasets' numeric columns differ, only the intersection is used; no error is raised. Because a correlation matrix is symmetric, only one triangle is returned (no duplicate var1/var2 vs. var2/var1 rows). The diagonal is dropped: a variable's correlation with itself is always 1 in both datasets, so its difference is always 0 and carries no information.

Value

A tibble with one row per shared variable pair, with columns `var1`, `var2`, and `diff` (comparison correlation minus baseline correlation).

Examples

```
set.seed(1)
baseline <- mtcars
comparison <- mtcars[sample(nrow(mtcars), replace = TRUE), ]
correlation_diff(baseline, comparison)
```

---

correlation_diff_heatmap

Heatmap of Correlation Differences Between Two Datasets

Description

Renders the output of `correlation_diff` as a heatmap (variable by variable), using the same visual language as `pca_loading_diff_heatmap`: a diverging fill scale centered at zero, so variable pairs with little change are white and larger correlation changes in either direction stand out in Temple Night Owl or cherry.

Usage

```
correlation_diff_heatmap(
  baseline_data,
  comparison_data,
  cor.use = "everything",
  cor.method = "pearson",
  na_omit = TRUE
)
```

Arguments

`baseline_data`	A data frame or tibble treated as the reference.
`comparison_data`	A data frame or tibble to compare against `baseline_data`.
`cor.use`, `cor.method`, `na_omit`	See `correlation_plot`.

Value

A ggplot object.

Examples

```
set.seed(1)
baseline <- mtcars
comparison <- mtcars[sample(nrow(mtcars), replace = TRUE), ]
correlation_diff_heatmap(baseline, comparison)
```

`correlation_plot` *Correlation Plot*

Description

Plots a correlation matrix across all numeric columns of `data` using **corrplot**.

Usage

```
correlation_plot(
  data,
  cor.use = "everything",
  cor.method = "pearson",
  method = "ellipse",
  type = "upper",
  order = "FPC",
  title = "Correlation Coefficient Plot",
  na_omit = TRUE,
  tl.cex = 0.5,
  number.cex = 0.75,
  tl.srt = 45,
  mar = c(0, 0, 2, 0),
  show_coef = TRUE,
  ...
)
```

Arguments

`data`	A data frame or tibble.
`cor.use`	Passed to `cor`: `"everything"` (default), `"all.obs"`, `"complete.obs"`, `"na.or.complete"`, or `"pairwise.complete.obs"`.
`cor.method`	Passed to `cor`: `"pearson"` (default), `"kendall"`, or `"spearman"`.
`method`	Visualization method passed to `corrplot` (default `"ellipse"`).
`type`	`"upper"` (default), `"full"`, or `"lower"`.
`order`	Ordering method for the correlation matrix (default `"FPC"`, first principal component order).
`title`	Plot title.
`na_omit`	Logical (default `TRUE`); drop rows with any `NA` among the numeric columns before computing correlations.
`tl.cex`, `number.cex`, `tl.srt`	Label/number sizing and rotation, passed to `corrplot`.
`mar`	Plot margin, passed to `corrplot` in the standard `par("mar")` form `c(bottom, left, top, right)` (default `c(0, 0, 2, 0)`). `corrplot()` does not otherwise reserve any extra space above the matrix for `title`, so with the default (zero) margin the title text collides with the 45-degree diagonal column labels sitting just below it; the default here adds two lines of top margin so the title clears the labels. Increase further if using a long title or a larger `tl.cex`.

show_coef — Logical (default TRUE); whether to draw the correlation coefficient inside each cell. On a small matrix the numbers add useful precision on top of the visual encoding, but on a large matrix (many variables) they quickly overlap each other and the diagonal labels. Set to FALSE to omit them entirely – this is the clean way to declutter a large matrix; shrinking tl.cex and number.cex down to near-zero to visually "hide" the numbers (as earlier callers did) also erases the variable name labels and should be avoided. See correlation_plot_split for an alternative that keeps coefficients readable by splitting a large matrix into several smaller plots instead of hiding them.

... — Additional arguments passed to corrplot.

Value

Invisibly, the correlation matrix (called for its plot side effect).

Examples

```
correlation_plot(mtcars, tl.cex = .7)
correlation_plot(mtcars, tl.cex = .7, show_coef = FALSE)
```

---

correlation_plot_split

Correlation Plot, Split Into Legible Sub-Plots

Description

Draws the same style of correlation matrix as correlation_plot, but for matrices with too many variables to stay legible in a single plot (for example, ~40 clinical parameters), it first groups variables via hierarchical clustering on their correlation structure and then draws one correlation_plot-style plot per group, showing only the within-group correlations.

Usage

```
correlation_plot_split(
  data,
  cor.use = "everything",
  cor.method = "pearson",
  na_omit = TRUE,
  group_size = 12,
  title = "Correlation Coefficient Plot",
  ...
)
```

Arguments

data — A data frame or tibble.

cor.use, cor.method, na_omit — See correlation_plot.

group_size — Target number of variables per sub-plot (default 12). The number of groups is ceiling(n_vars / group_size); groups may end up somewhat smaller or larger than this target since cutree produces clusters of whatever sizes the dendrogram structure dictates, not exactly equal-sized groups.

`title`	Base plot title; each sub-plot's title has `" (Group i of n)"` appended so the sub-plots can be told apart.
`...`	Additional arguments passed on to `correlation_plot` for each sub-plot (e.g. `tl.cex`, `show_coef`, `method`).

Details

Variables are clustered on `as.dist(1 - abs(cor_mat))` – the same correlation-based distance `corrplot`'s own `order = "hclust"` uses internally – so that variables which move together end up in the same sub-plot instead of being split arbitrarily (e.g. alphabetically). The resulting dendrogram is cut into `ceiling(n_vars / group_size)` contiguous groups via `cutree`.

The default `group_size` of 12 is chosen to match `correlation_plot`'s own defaults: at the default `tl.cex`/`number.cex`, a matrix of roughly a dozen variables is about as many as can fit one legible diagonal label and coefficient per cell without crowding – the same order of magnitude as the small examples (like `mtcars`'s 11 numeric columns) `correlation_plot()` was originally tuned against. Larger or smaller groups can be requested via `group_size` depending on label length and plot size.

Value

Invisibly, a named list of per-group correlation matrices (one matrix per sub-plot, named `"Group 1"`, `"Group 2"`, ...).

Examples

```
# A data frame with more numeric columns than fit legibly in one plot.
wide_data <- cbind(mtcars, iris[seq_len(nrow(mtcars)), sapply(iris, is.numeric)])
correlation_plot_split(wide_data, group_size = 6)
```

---

corr_test_all *Pairwise Correlation Tests Across All Numeric Columns*

Description

Runs `cor.test` on every pair of numeric columns in a data frame and returns one row per pair.

Usage

```
corr_test_all(
  data,
  method = "pearson",
  use = c("pairwise.complete.obs", "complete.obs"),
  columns = c("compact", "tidy"),
  sort = c("p_value", "estimate", "abs_estimate", "none"),
  ...
)
```

Arguments

data	A data frame or tibble. Non-numeric columns are ignored.
method	Correlation method passed to cor.test: "pearson" (default), "kendall", or "spearman".
use	How missing values are handled. "pairwise.complete.obs" (default) tests each pair on the rows where both columns are observed, which is what cor.test() does natively. "complete.obs" first drops every row with a missing value in any numeric column, so all pairs are tested on the same rows.
columns	Output shape. "compact" (default) returns var1, var2, r, p_value. "tidy" returns var1, var2, then every column broom::tidy() reports for the test, with estimate renamed cor: cor, statistic, p.value, parameter, conf.low, conf.high, method, alternative. parameter and the confidence limits are only reported by methods that compute them (Pearson).
sort	Row order. "p_value" (default) is ascending p-value; "estimate" is descending correlation; "abs_estimate" is descending absolute correlation; "none" keeps pair order. Ties keep pair order, and NAs sort last.
...	Further arguments passed to cor.test, such as alternative, conf.level, or exact.

Details

Pairs are enumerated in column order (via combn), so var1 is always the column that appears first in data. A pair whose test cannot be computed (e.g. fewer than three complete observations) is kept with NA results rather than aborting the whole run. Invalid arguments (an unknown method, alternative, a bad conf.level, ...) are errors, not NA rows.

Value

A tibble with one row per pair of numeric columns.

Examples

```
corr_test_all(iris[, 1:4])

# Every cor.test() statistic, strongest positive correlation first
corr_test_all(mtcars[, c("mpg", "hp", "wt", "qsec")], columns = "tidy", sort = "estimate")
```

---

coxnet *Penalized Cox Regression for Right-Censored or Start/Stop Survival Data*

Description

Fits an elastic-net penalized Cox proportional hazards model with [glmnet::glmnet()], through the tidymodels **hardhat** interface: a formula, a recipe, or predictors and outcome given separately. The outcome is a [survival::Surv()] object, either right-censored, 'Surv(time, event)', or counting-process, 'Surv(start, stop, event)', for time-varying covariates.

Usage

```
coxnet(x, ...)

## Default S3 method:
coxnet(x, ...)

## S3 method for class 'data.frame'
coxnet(x, y, penalty = NULL, mixture = 1, path = NULL, ...)

## S3 method for class 'matrix'
coxnet(x, y, penalty = NULL, mixture = 1, path = NULL, ...)

## S3 method for class 'formula'
coxnet(formula, data, penalty = NULL, mixture = 1, path = NULL, ...)

## S3 method for class 'recipe'
coxnet(x, data, penalty = NULL, mixture = 1, path = NULL, ...)
```

Arguments

x	A data frame or matrix of predictors, or a [recipes::recipe()] whose outcome is a single ‘Surv‘ column.
...	Further arguments passed to [glmnet::glmnet()], such as ‘nlambda‘ or ‘cox.ties‘. Not ‘x‘, ‘y‘, ‘family‘, ‘alpha‘, or ‘lambda‘. ‘cox.ties‘ defaults to ‘"breslow"‘, matching the Breslow baseline hazard used for survival predictions, so results don’t depend on glmnet’s own default (Breslow in glmnet 5.0, Efron from 5.1).
y	A [survival::Surv()] object with one element per row of ‘x‘.
penalty	Default penalty (glmnet’s ‘lambda‘) used by ‘predict()‘ and ‘tidy()‘. A single non-negative number, or ‘NULL‘ to require it there. glmnet’s default path can end above a small ‘penalty‘ (it stops early once the deviance ratio saturates), so when ‘penalty‘ is below the path it is continued down to ‘penalty‘; ‘0‘ gives the unpenalized fit. Penalties between path values are interpolated, as by [glmnet::predict.glmnet()].
mixture	Elastic-net mixing parameter (glmnet’s ‘alpha‘): 1 is the lasso, 0 is ridge regression.
path	Optional decreasing sequence of penalties to fit, instead of glmnet’s default path. It is used as given: ‘predict()‘ and ‘tidy()‘ at a penalty below its smallest value warn and use the smallest.
formula	A formula with a ‘Surv()‘ outcome, such as ‘survival::Surv(tstart, tstop, status) ~ age + sex‘.
data	A data frame containing the variables in ‘formula‘ or used by the recipe.

Details

The whole regularization path is fit, as glmnet recommends; ‘penalty‘ only sets the default penalty for [predict()][predict.coxnet_model()] and [tidy()][tidy.coxnet_model()]. To choose ‘penalty‘ and ‘mixture‘ by cross-validation, use [cv_coxnet()].

Predictors must be numeric, and finite: missing values and ‘Inf‘/‘-Inf‘ in the predictors or the follow-up times are an error, because glmnet would give an infinite predictor a coefficient of 0 without a word. The formula interface expands factors into one indicator column per level, which suits

penalized models; with a recipe, add ‘recipes::step_dummy()‘. In a formula, write ‘survival::Surv()‘ unless **survival** is attached.

‘strata()‘ and ‘offset()‘ are not supported in a formula, here or in [cv_coxnet()] and [nested_cv_coxnet()]: ‘strata()‘ would be fit as ordinary penalized indicator columns, with no stratified baseline hazard, and ‘offset()‘ would be dropped, so the model would not be the one the formula describes. A formula that uses either stops with an error; use [survival::coxph()] for stratified models and offsets.

Value

A ‘coxnet_model‘ object: the glmnet fit, ‘penalty‘, ‘mixture‘, the training predictor matrix ‘x‘ and outcome ‘y‘ (kept so survival can be predicted at any penalty), and the hardhat ‘blueprint‘.

See Also

[predict.coxnet_model()], [tidy.coxnet_model()], [cv_coxnet()], [nested_cv_coxnet()]

Examples

```
if (requireNamespace("glmnet", quietly = TRUE) &&
    requireNamespace("survival", quietly = TRUE)) {
 lung <- stats::na.omit(survival::lung[, c("time", "status", "age", "sex", "ph.ecog", "wt.loss")])
 fit <- coxnet(survival::Surv(time, status) ~ ., data = lung, penalty = 0.05, mixture = 0.5)
  generics::tidy(fit)
  predict(fit, lung[1:3, ], type = "survival", eval_time = c(180, 365))$.pred[[1]]
}
```

---

coxnet_train *The ‘coxnet‘ Engine for ‘parsnip::proportional_hazards()‘*

Description

Registers [coxnet()] as an engine of [parsnip::proportional_hazards()] in ‘"censored regression"‘ mode, so penalized Cox models fit by TempleCBE work anywhere a parsnip model does: [parsnip::fit()], **workflows**, **tune**, **workflowsets**, and **stacks**. Unlike **censored**’s ‘"glmnet"‘ engine, it also accepts counting-process outcomes, ‘Surv(start, stop, event)‘, for time-varying covariates.

Usage

```
coxnet_train(
  formula,
  data,
  penalty = NULL,
  mixture = NULL,
  weights = NULL,
  ...
)

predict_coxnet_linear_pred(object, new_data, penalty = NULL)

predict_coxnet_time(object, new_data, penalty = NULL)

predict_coxnet_survival(object, new_data, eval_time, penalty = NULL)
```

Arguments

formula	A formula with a [survival::Surv()] outcome.
data	A data frame containing the variables in ‘formula‘.
penalty	The penalty, glmnet’s ‘lambda‘: a single non-negative number.
mixture	Elastic-net mixing parameter, glmnet’s ‘alpha‘. ‘NULL‘ means 1.
weights	Not supported; must be ‘NULL‘.
...	Further arguments passed to [glmnet::glmnet()].
object	A parsnip ‘model_fit‘ using the coxnet engine.
new_data	A data frame of new predictors.
eval_time	Times at which to predict survival.

Details

The engine is registered when **parsnip** is loaded, whichever package is loaded first; nothing needs to be called.

Use it like any other engine:

“‘r spec <- parsnip::proportional_hazards(penalty = 0.05, mixture = 0.5) |> parsnip::set_engine("coxnet") fit <- parsnip::fit(spec, survival::Surv(time, status) ~ ., data = lung) predict(fit, lung, type = "linear_pred") predict(fit, lung, type = "survival", eval_time = c(180, 365)) “‘

* ‘penalty‘ is required (it is glmnet’s ‘lambda‘); ‘mixture‘ defaults to 1, the lasso. Both can be [tune::tune()]d. Each candidate is fit separately; the coxnet engine doesn’t use glmnet’s path to evaluate a grid of penalties from one fit, so tuning ‘penalty‘ alone costs one fit per value. * Engine arguments given to [parsnip::set_engine()] are passed on to [glmnet::glmnet()], such as ‘nlambda‘ or ‘cox.ties‘. * Predictions: ‘"linear_pred"‘ (larger means longer survival, as in **censored**), ‘"survival"‘, which needs ‘eval_time‘, and ‘"time"‘, the restricted mean survival time, which static metrics such as [yardstick::concordance_survival()] use. * Case weights aren’t supported, and neither is ‘strata()‘ in the formula. * [generics::tidy()] on the parsnip fit returns the coefficients.

Value

For ‘coxnet_train()‘, a [coxnet()] model. This is the function parsnip calls to fit the engine; call [coxnet()] directly outside parsnip.

See Also

[coxnet()], [cv_coxnet()], [parsnip::proportional_hazards()], ‘censored::proportional_hazards‘ engines

Examples

```
if (requireNamespace("parsnip", quietly = TRUE) &&
    requireNamespace("glmnet", quietly = TRUE) &&
    requireNamespace("survival", quietly = TRUE)) {
  lung <- stats::na.omit(survival::lung[, c("time", "status", "age", "sex", "ph.ecog")])
  spec <- parsnip::set_engine(
    parsnip::proportional_hazards(penalty = 0.05, mixture = 0.5),
    "coxnet"
  )
  fit <- parsnip::fit(spec, survival::Surv(time, status) ~ ., data = lung)
  predict(fit, lung[1:3, ], type = "survival", eval_time = c(180, 365))
}
```

create_qmd_renderer *Create a FileOutputs object for a rendering QMD or R Markdown document*

Description

Create a FileOutputs object for a rendering QMD or R Markdown document

Usage

```
create_qmd_renderer(
  name,
  path,
  deps = list(),
  file_stage = NA_character_,
  output_format = getOption("pipeline.default_formats", "pdf"),
  description = NA_character_
)
```

Arguments

name	The nickname for the file.
path	The full path to the .qmd or .Rmd file.
deps	A list of dependency objects (e.g., list(og_DATA)).
file_stage	The 'stage' for this file.
output_format	Output format(s): e.g. "pdf", "html", c("pdf", "docx"), or "yaml" to automatically extract the output formats declared in the document's YAML frontmatter. Defaults to getOption("pipeline.default_formats", "pdf"). Each output file gets the extension its format writes, which is not always the format's name: revealjs, ioslides, slidy and html_document write .html, beamer and typst write .pdf, gfm and github_document write .md. A format added by an extension (<extension>-<base format>, e.g. temple-html) writes its base format's file. For a format whose extension is not known the format name is used, with a warning.
description	A brief summary of the document.

Value

A FileOutputs object.

create_report *Scaffold a New Report From a Template*

Description

Copies a bundled report template (and its supporting bibliography/title files) into location.

Usage

```
create_report(
  location = getwd(),
  template_name = "t_test_example",
  child = TRUE,
  type = ".qmd",
  include_bib = TRUE,
  include_tex = TRUE,
  install_brand = FALSE,
  filename = NULL
)
```

Arguments

location	Directory to create the report in (default getwd()).
template_name	One of "t_test_example" (default), "example", or "temple".
child	Logical (default TRUE); also copy the child-document template.
type	One of ".qmd" (default) or ".Rmd".
include_bib	Logical (default TRUE); also copy the .bib file.
include_tex	Logical (default TRUE); also copy the title .tex file.
install_brand	Logical (default FALSE); for the "temple" template, also run use_temple_brand(location), which downloads the extension into location itself. Leave this FALSE when the extension is already installed at the project root (see use_temple_brand); it doesn't need reinstalling per report.
filename	Base name (no extension) for the report file, default NULL uses template_name. The report is always written to location, so calling create_report() twice for the same location with the same template_name (or the same filename) overwrites the first report; pass a distinct filename for each report that shares a location, e.g. create_report("analysis", filename = "analysis1") and create_report("analysis", filename = "analysis2").

Details

The "temple" template renders with the quarto_temple_brand extension's temple-html, temple-pdf, and temple-typst formats, so it needs that extension installed somewhere Quarto can find it from location (see use_temple_brand): either at an ancestor project root shared by every report, or, with install_brand = TRUE, in location itself for a one-off report. It uses no title.tex or child document, and is Quarto-only.

Value

A list indicating whether each file was created.

Examples

```
## Not run:
create_report(here::here("analysis"))

# A project with several "temple" reports: install the extension once at
# the project root, then scaffold each report without install_brand.
use_temple_brand(here::here())
```

```
create_report(here::here("analysis"), template_name = "temple")
create_report(here::here("reports"), template_name = "temple", filename = "q3")

# A single one-off "temple" report instead installs beside itself.
create_report(here::here("analysis"), template_name = "temple", install_brand = TRUE)

# Two reports sharing one location need distinct `filename`s, or the
# second call overwrites the first report file:
create_report(here::here("analysis"), template_name = "temple", filename = "analysis1")
create_report(here::here("analysis"), template_name = "temple", filename = "analysis2")

## End(Not run)
```

`create_toc_from_sas_pdf`

Build a Table of Contents from a SAS-Generated PDF

Description

Scans a PDF (as produced by SAS reporting output) for table titles and writes a Table-of-Contents `.rtf` file alongside it, prefixed `TOC_`.

Usage

```
create_toc_from_sas_pdf(input, collapse = FALSE, top_margin_height = 60)
```

Arguments

`input`	Path to the input PDF.
`collapse`	If `FALSE` (default), every occurrence of a title is listed; if `TRUE`, only the first in a numbered sequence is kept.
`top_margin_height`	How far down the page (in PDF text-position units) to look for titles.

Value

Invisibly, a message string naming the output file path.

Examples

```
## Not run:
create_toc_from_sas_pdf("path/to/report.pdf", collapse = TRUE)

## End(Not run)
```

cv_coxnet *Cross-Validate a Penalized Cox Model With yardstick Survival Metrics*

Description

A tidymodels counterpart to [glmnet::cv.glmnet()] for right-censored and start/stop survival data. Every combination of ‘mixture‘ and ‘penalty‘ is fit on each analysis set and scored on the assessment set with a **yardstick** survival metric set, and the best settings are chosen by one metric: the integrated Brier score by default.

Usage

```
cv_coxnet(x, ...)

## Default S3 method:
cv_coxnet(x, ...)

## S3 method for class 'data.frame'
cv_coxnet(
  x,
  y,
  subject_id = NULL,
  group = NULL,
  mixture = 1,
  penalty = NULL,
  resamples = NULL,
  v = 10,
  metrics = NULL,
  eval_time = NULL,
  metric = NULL,
  covariates = c("path", "baseline"),
  trunc = 0.05,
  parallel = FALSE,
  check_subject_overlap = TRUE,
  ...
)

## S3 method for class 'matrix'
cv_coxnet(
  x,
  y,
  subject_id = NULL,
  group = NULL,
  mixture = 1,
  penalty = NULL,
  resamples = NULL,
  v = 10,
  metrics = NULL,
  eval_time = NULL,
  metric = NULL,
  covariates = c("path", "baseline"),
```

```
  trunc = 0.05,
  parallel = FALSE,
  check_subject_overlap = TRUE,
  ...
)

## S3 method for class 'formula'
cv_coxnet(
  formula,
  data,
  subject_id = NULL,
  group = NULL,
  mixture = 1,
  penalty = NULL,
  resamples = NULL,
  v = 10,
  metrics = NULL,
  eval_time = NULL,
  metric = NULL,
  covariates = c("path", "baseline"),
  trunc = 0.05,
  parallel = FALSE,
  check_subject_overlap = TRUE,
  ...
)

## S3 method for class 'recipe'
cv_coxnet(
  x,
  data,
  subject_id = NULL,
  group = NULL,
  mixture = 1,
  penalty = NULL,
  resamples = NULL,
  v = 10,
  metrics = NULL,
  eval_time = NULL,
  metric = NULL,
  covariates = c("path", "baseline"),
  trunc = 0.05,
  parallel = FALSE,
  check_subject_overlap = TRUE,
  ...
)
```

Arguments

x	A data frame or matrix of predictors, or a [recipes::recipe()] whose outcome is a single ‘Surv‘ column.
...	Further arguments passed to [glmnet::glmnet()], such as ‘cox.ties‘ or ‘nlambda‘.
y	For the data frame and matrix methods, a [survival::Surv()] outcome with one

element per row of ‘x‘.

`subject_id` Subject identifier: a column name for the formula and recipe methods (for a recipe, a single column with role ‘"id"‘ is used when ‘NULL‘), or a vector for the data frame and matrix methods. Required for start/stop data; for right-censored data, each row is a subject when ‘NULL‘.

`group` Optional grouping for folds, in the same form as ‘subject_id‘. Defaults to ‘subject_id‘.

`mixture` Numeric vector of elastic-net mixing values to try.

`penalty` Optional decreasing penalty path; by default glmnet's path for each ‘mixture‘.

`resamples` An ‘rset‘ of ‘data‘, such as [rsample::group_vfold_cv()]. Overrides ‘v‘ and ‘group‘. No subject (or group) may be in both the analysis and the assessment set of a split; see Details and ‘check_subject_overlap‘.

`v` Number of folds when ‘resamples‘ is ‘NULL‘.

`metrics` A [yardstick::metric_set()] of survival metrics. Defaults to the integrated Brier score, concordance, time-specific Brier score, and time-dependent ROC AUC.

`eval_time` Evaluation times. Defaults to deciles (10th to 90th percentile) of the observed event times.

`metric` Name of the metric used to choose settings; the first metric in ‘metrics‘ by default.

`covariates` ‘"path"‘ or ‘"baseline"‘; see Details.

`trunc` Lower bound for the censoring probability in the Graf weights.

`parallel` If ‘TRUE‘, fit folds with [furrr::future_map()]; set a [future::plan()] first.

`check_subject_overlap`
If ‘TRUE‘ (the default), stop when a supplied ‘resamples‘ puts the same subject (or, when ‘group‘ is set, the same group) in both the analysis and the assessment set of a split; see Details. ‘FALSE‘ skips the check. Folds that ‘cv_coxnet()‘ builds itself are never checked, as they are grouped by subject already.

`formula` A formula with a ‘Surv()‘ outcome, as in [coxnet()].

`data` A data frame with the outcome, predictors, ‘subject_id‘, and ‘group‘ columns.

Details

Folds. Unless ‘resamples‘ is given, folds come from [rsample::group_vfold_cv()] grouped by ‘group‘ (default: ‘subject_id‘), so a subject's start/stop rows are never split between analysis and assessment sets, as they would be by ‘cv.glmnet()‘'s row-level folds. Group by a coarser unit, such as study site, for leave-sites-out cross-validation; each subject must then belong to one group.

Supplied resamples. With ‘resamples‘, every split is checked before anything is scored: if a subject (or, when ‘group‘ is set, a group) is in both the analysis and the assessment set of a split, ‘cv_coxnet()‘ stops, because the model would be scored on subjects it was fit on. That is what a row-level [rsample::vfold_cv()] does to start/stop data. Group the folds by the subject instead, with [rsample::group_vfold_cv()] or [rsample::group_bootstraps()]; the out-of-bag assessment set of a bootstrap never overlaps its analysis set. ‘check_subject_overlap = FALSE‘ switches the check off, for overlap that is deliberate.

Fitting. For each ‘mixture‘, one penalty path is computed on all of ‘data‘ (as ‘cv.glmnet()‘ does) and every fold is fit along it. With a recipe, the recipe is prepped on each analysis set, so preprocessing is learned inside the fold.

Predictions. Survival for each assessment-set subject is computed from the Breslow baseline hazard of the analysis set. With ‘covariates = "path"‘, the cumulative hazard is integrated over the

subject's start/stop covariate path, carrying each interval's values forward to the next interval and past the last one; with ‘"baseline"‘, only the covariates of the first interval are used, which uses no information from after time 0. The static concordance metric uses predicted survival at the last ‘eval_time‘.

Scoring. Start/stop rows are collapsed to one row per subject ([surv_subject_truth()]), weighted for censoring with Graf weights from the analysis set ([graf_weights()]), and passed to ‘metrics‘. An assessment-set subject with a missing predictor after preprocessing, such as a factor level the analysis set never saw, cannot be predicted: it is left out at every penalty, so all penalties are scored on the same subjects, with one warning per resample that gives the count. A resample with no subject left is skipped.

Selection. ‘lambda.min‘ is the penalty, across all ‘mixture‘ values, with the best mean ‘metric‘ over folds (for time-specific metrics, at the first ‘eval_time‘). ‘lambda.1se‘ is the largest penalty, at the same mixture, whose mean is within one standard error of that best mean.

Value

A ‘cv_coxnet‘ object, with: * ‘metrics‘: mean, ‘n‘, and ‘std_err‘ of each metric by ‘mixture‘, ‘penalty‘, and ‘.eval_time‘; * ‘fold_metrics‘: every fold's metrics; * ‘best_by_mixture‘: ‘lambda_min‘ and ‘lambda_1se‘ for each ‘mixture‘; * ‘mixture‘, ‘lambda_min‘, ‘lambda_1se‘, ‘metric‘, ‘direction‘; * ‘fit‘: a [coxnet()] model fit to all of ‘data‘ at the chosen ‘mixture‘, with ‘lambda_min‘ as its default penalty; * ‘eval_time‘, ‘resamples‘, and ‘covariates‘.

Use ‘predict()‘, ‘tidy()‘, ‘autoplot()‘, and ‘tune::collect_metrics()‘ on it.

See Also

[coxnet()], [nested_cv_coxnet()], [surv_subject_truth()]

Examples

```
if (requireNamespace("glmnet", quietly = TRUE) &&
    requireNamespace("survival", quietly = TRUE) &&
    requireNamespace("rsample", quietly = TRUE) &&
    requireNamespace("yardstick", quietly = TRUE)) {
  set.seed(1)
  # Start/stop data: up to 3 visits per subject, events only at the last one
  long <- do.call(rbind, lapply(1:80, function(i) {
    k <- sample(1:3, 1)
    stops <- cumsum(stats::runif(k, 2, 6))
    risk <- stats::rnorm(1)
    data.frame(subject_id = i, tstart = c(0, utils::head(stops, -1)), tstop = stops,
               status = c(rep(0, k - 1), stats::rbinom(1, 1, stats::plogis(risk))),
          x1 = risk + stats::rnorm(k, sd = 0.2), x2 = stats::rnorm(k), x3 = stats::rnorm(k))
  }))
  cv <- cv_coxnet(
    survival::Surv(tstart, tstop, status) ~ x1 + x2 + x3,
    data = long, subject_id = "subject_id", mixture = c(0.5, 1),
    v = 5, nlambda = 20, cox.ties = "breslow"
  )
  cv
  generics::tidy(cv, penalty = "lambda.1se")
}
```

`cv_coxnet-methods` *Use a 'cv_coxnet' Result*

Description

Predict from, tidy, plot, or collect the metrics of a [cv_coxnet()] result. Predictions and coefficients come from the model refit to all the data at the chosen 'mixture'.

Usage

```
## S3 method for class 'cv_coxnet'
predict(
  object,
  new_data,
  type = c("linear_pred", "survival"),
  penalty = "lambda.min",
  eval_time = NULL,
  increasing = TRUE,
  ...
)

## S3 method for class 'cv_coxnet'
tidy(x, penalty = "lambda.min", ...)

## S3 method for class 'cv_coxnet'
collect_metrics(x, ..., summarize = TRUE)

## S3 method for class 'cv_coxnet'
autoplot(object, ...)
```

Arguments

`object, x`	A [cv_coxnet()] result.
`new_data`	A data frame of new predictors.
`type`	'"linear_pred"' (the default), '"survival"', or '"time"' (restricted mean survival time), as for [predict.coxnet_model()].
`penalty`	'"lambda.min"' (default), '"lambda.1se"', or a number.
`eval_time, increasing`	As for [predict.coxnet_model()].
`...`	Not used.
`summarize`	For 'collect_metrics()': 'TRUE' for means over resamples, 'FALSE' for each resample's metrics.

Value

'predict()': a tibble as from [predict.coxnet_model()]. 'tidy()': the coefficients at 'penalty'. 'collect_metrics()': a tibble of metrics. 'autoplot()': a ggplot of the selection metric against the penalty, with 'lambda.min' (solid) and 'lambda.1se' (dashed) marked.

cv_joint_model *Cross-Validation for Joint Models*

Description

Evaluates the joint_model trio across cross-validation splits (e.g. via vfold_cv or group_vfold_cv), benchmarking survival, classification, and regression paradigms using the IPCW Integrated Brier Score (the yardstick::brier_survival_integrated() value, as cv_coxnet and glmnet_IBS report it) and Concordance. Each fold's model is fit on the analysis set and scored on the assessment set, one row per subject, with Graf censoring weights from the analysis set.

Usage

```
cv_joint_model(
  data,
  outcome = survival::Surv(time, status) ~ .,
  v = 5,
  resamples = NULL,
  subject_id = NULL,
  engine = c("glmnet", "baguette", "stacks"),
  calibration = TRUE,
  parallel = FALSE,
  covariates = c("path", "baseline"),
  check_subject_overlap = TRUE,
  ...
)
```

Arguments

data	A data frame.
outcome	Survival formula with a Surv() outcome.
v	Number of cross-validation folds (default 5).
resamples	Optional pre-constructed **rsample** object. No subject may be in both the analysis and the assessment set of a split; see Details and check_subject_overlap.
subject_id	Subject identifier column. Required for start/stop outcomes; used for grouped resampling.
engine	Modeling engine: "glmnet", "baguette", or "stacks" (the "baguette" models plus a stored Cox meta-learner whose output predict() adds as extra columns; see joint_model).
calibration	Logical; whether to calibrate status predictions (default TRUE).
parallel	Logical; whether to run folds in parallel via **furrr**.
covariates	"path" or "baseline"; see Details. The same as cv_coxnet's argument, and also used when penalty is NULL to score the cross-validation that tunes each fold's coxnet penalty.
check_subject_overlap	If TRUE (the default), stop when a supplied resamples puts the same subject in both the analysis and the assessment set of a split; see Details. FALSE skips the check. Folds that cv_joint_model() builds itself are never checked, as they are grouped by subject already.
...	Additional arguments passed to joint_model.

Details

Subjects. A counting-process (start/stop) outcome needs `subject_id`, so that a subject's intervals stay in one fold and are scored together; without it the function stops. For right-censored data each row is a subject when `subject_id` is `NULL`. Unless `resamples` is given, folds come from `group_vfold_cv` grouped by `subject_id`.

Supplied resamples. With `resamples`, every split is checked before anything is fit: if a subject is in both the analysis and the assessment set of a split, the function stops, because the models would be scored on subjects they were fit on. That is what a row-level `vfold_cv` does to start/stop data. Group the folds by the subject instead, with `group_vfold_cv` or `group_bootstraps`; the out-of-bag assessment set of a bootstrap never overlaps its analysis set. `check_subject_overlap = FALSE` switches the check off, for overlap that is deliberate.

Scoring start/stop data. The three models are scored on the same subjects, but not from the same rows:

- The `coxnet` model is scored as `cv_coxnet` scores it: the survival of each assessment subject comes from the Breslow baseline hazard of the analysis set and the subject's covariates. With `covariates = "path"` the cumulative hazard is integrated over the subject's start/stop covariate path, carrying each interval's values forward to the next interval and past the last one; with `"baseline"`, only the covariates of the first interval are used, which uses no information from after time 0. For the same fit, folds and `eval_time`, the `coxnet` IBS and concordance are those of `cv_coxnet()`. (Concordance uses the predicted survival at the last `eval_time`.)
- The status and time models predict one value per row, so for a subject they can only use one row. They use the **baseline** (first-interval) row, whatever `covariates` is.

Missing predictors. An assessment subject with a missing predictor after preprocessing, such as a factor level the analysis set never saw, cannot be predicted by any of the models: it is left out of the scores of all three, with one warning per resample that gives the count. A resample with no subject left is skipped.

Value

An S3 object of class `c("cv_joint_model", "tbl_df")` summarizing comparative metrics across folds: `model`, `ibs`, `concordance`, and `fold`.

`database_metadata_io` *Export or Import Dataset Metadata*

Description

Exports and imports data dictionaries/metadata to and from `.csv` or `.xlsx` formats for in-place review and editing.

Usage

```
write_database_metadata(metadata, path, ...)

read_database_metadata(path, sheet = NULL, ...)
```

Arguments

metadata	A tibble/data.frame of dataset metadata (e.g. from get_dataset_info).
path	File path ending in .csv or .xlsx.
...	Additional arguments passed to the underlying writer/reader.
sheet	Optional sheet name when reading from Excel (defaults to 1 or "METADATA").

Value

For writers, path invisibly. For readers, a tibble of metadata.

---

database_setops *Set Operations on R Databases*

Description

cbe_database methods for the generic dplyr::intersect()/dplyr::union()/dplyr::setdiff() verbs (the same generics **dbplyr/dtplyr** extend for their own backends) – the table-level building blocks behind graph_intersect/graph_subtract/ graph_union. Given two as_database results (or any plain list(nodes = ..., edges = ...) passed through as_database() first), these find their common, exclusive, or combined rows. Operating at this level (rather than only on graph objects) means they also work directly on nodes/edges tables that never came from a graph – e.g. two tables read back from read_workbook.

Arguments

x, y	cbe_database objects (see as_database): named lists with nodes and edges data frames/tibbles.
node_by	Column(s) to match nodes on. NULL (default) uses the first column of each nodes table. A single string uses that column name on both sides. A named vector/string c(x_col = y_col) matches x's x_col to y's y_col (as in dplyr::*_join()).
edge_by	Column(s) to match edges on (default c("from", "to"), which every as_database method produces).
...	Passed on; unused.

Details

Dispatch is on the cbe_database class (stamped by as_database), not on igraph – **igraph** already registers its own union.igraph() on this same generic, and S3 method tables are global, so a union.igraph() defined here would silently collide with it. graph_intersect/graph_subtract/graph_union stay dedicated functions for exactly this reason, built on top of these cbe_database methods via as_database/database_to_igraph.

Node matching defaults to the first column of each database's nodes table – the node identifier column, which different sources name differently (name for an igraph-derived database, id for a **visNetwork**/funviewR one). When the two databases use different column names, pass node_by explicitly using the same c(x_col = y_col) convention as dplyr::*_join(by = ...); a plain string means the same column name on both sides.

intersect()/setdiff()**: attribute values always come from** x**, never** y**, and are never merged.** dplyr::semi_join()/ dplyr::anti_join() filter x's rows by whether a key matches in y – they

don't pull any of y's own columns across. This matters most for graphs whose nodes carry rich, node-specific metadata – e.g. a pipeline graph built from FilePath/FileUses/ FileOutputs objects (see compute_graph), where stage, mtime, and description differ node by node. Intersecting two snapshots of the same pipeline graph (say, today's vs. yesterday's) for a node present in both keeps *today's* mtime/stage, silently discarding yesterday's – there is no reconciliation between the two. To compare attribute values themselves (not just which nodes exist), run cbe_compare_df on the two as_database results directly instead.

union()**: attribute values are coalesced, not taken from one side only.** For a node/edge present in both x and y, shared columns keep x's value where it's non-NA and fall back to y's otherwise – so combining two databases never silently drops data the way intersect()/setdiff() can. This mirrors join_pipelines's existing union-and-coalesce behavior for tbl_graph objects, generalized to any as_database source.

Value

A cbe_database list(nodes = <tibble>, edges = <tibble>). For intersect()/setdiff(), edges are always constrained to reference only the returned nodes, so the result is a well-formed graph database even if a matched/unmatched edge's endpoint didn't independently match on the node side.

One consequence of that well-formedness constraint: an edge can vanish from *both* intersect() and setdiff() at once. If edge b->c doesn't literally match one of y's edges, it's excluded from the intersection – but if node b is itself one of y's nodes, b (and anything touching it) also drops out of the setdiff result too, taking b->c with it. The edge belongs to neither side; it isn't double-dropped or double-counted, just genuinely unrepresentable in a well-formed result for either operation.

Examples

```
db_x <- as_database(list(
  nodes = data.frame(name = c("a", "b", "c")),
  edges = data.frame(from = c("a", "b"), to = c("b", "c"))
))
db_y <- as_database(list(
  nodes = data.frame(name = c("a", "b")),
  edges = data.frame(from = "a", to = "b")
))
dplyr::intersect(db_x, db_y)
dplyr::setdiff(db_x, db_y)
dplyr::union(db_x, db_y)

# intersect()/setdiff(): attribute values come from x only, never merged
# with y's. Node "a" is present in both snapshots below with different
# `stage`/`mtime` (mirroring FilePath's own slots, see compute_graph()) --
# the intersection keeps today's values, not yesterday's.
snapshot_today <- as_database(list(
  nodes = data.frame(
    name = "a", stage = "02_analysis",
    mtime = as.POSIXct("2026-09-18")
  ),
  edges = data.frame(from = character(0), to = character(0))
))
snapshot_yesterday <- as_database(list(
  nodes = data.frame(
    name = "a", stage = "01_eda",
    mtime = as.POSIXct("2026-09-17")
  ),
```

```
  edges = data.frame(from = character(0), to = character(0))
))
dplyr::intersect(snapshot_today, snapshot_yesterday)$nodes$stage # "02_analysis"

# union(): coalesces instead -- yesterday's mtime fills in where a node
# only exists in one snapshot, and today's non-NA values win on overlap.
dplyr::union(snapshot_today, snapshot_yesterday)$nodes
```

`database_to_igraph` *Build an igraph From an R Database's Nodes/Edges Tables*

Description

The other direction from `as_database`: takes a `cbe_database` (or plain `list(nodes = ..., edges = ...)`) – such as one produced by `as_database` or the `database_setops` verbs – and builds an `igraph` from it.

Usage

```
database_to_igraph(db, directed = TRUE)
```

Arguments

`db`	A `cbe_database` (or plain named list with `nodes` and `edges` data frames/tibbles), where `nodes`'s first column is the node identifier used by `edges`'s `from`/`to` columns.
`directed`	Logical, whether the graph is directed (default `TRUE`).

Details

Deliberately not a new `as_igraph()` method: `as_igraph.default()` (see `as_igraph`) already dispatches on plain `list` objects for its own, unrelated purpose (a list of `FilePath`/`FileUses`/`FileOutputs` pipeline objects); adding `as_igraph.list()` here would silently intercept every one of those calls instead of falling through to `.default`.

Value

An `igraph` object.

Examples

```
db <- as_database(list(
  nodes = data.frame(name = c("a", "b", "c")),
  edges = data.frame(from = c("a", "b"), to = c("b", "c"))
))
database_to_igraph(db)
```

delete_nul_files *Delete Stray 'nul' Files*

Description

Windows-only. `knitr` occasionally leaves behind a file literally named `nul` as a side effect of redirecting output to the Windows `NUL` device. This deletes files whose *basename* is exactly `"nul"` (case-insensitive) under `path`.

Usage

```
delete_nul_files(
  path = here::here(),
  .dontask = FALSE,
  .verify_command = FALSE
)
```

Arguments

`path`	Directory to search, defaults to `here::here()`.
`.dontask`	Logical (default `FALSE`); skip the confirmation prompt.
`.verify_command`	Logical (default `FALSE`); if `TRUE`, return the paths that would be deleted, in device-namespace form, and delete nothing.

Details

The files are deleted from R, through their Windows device-namespace path (a plain path to such a file is taken for the `NUL` device itself), not by handing a command to `cmd.exe`, so a folder name that holds percent signs, such as `x%TEMP%y`, is safe. Folders that are reached through a link (a junction or symbolic link inside `path` that points elsewhere) are skipped, so nothing outside `path` is deleted.

Value

Invisibly, the deleted file paths (or, if `.verify_command = TRUE`, the paths that would be deleted). An error lists any file that could not be deleted.

demo_cbe_mapping *Create Sample Plug-and-Play Mapping Dictionary*

Description

Generates a demonstration column mapping table for documentation and testing.

Usage

```
demo_cbe_mapping()
```

Value

A tibble with sample mapping entries.

detect_outliers *Detect Outliers Across a Data Frame's Numeric Columns*

Description

Runs flag_outliers on every numeric column of data.

Usage

```
detect_outliers(data, outliers_only = TRUE)
```

Arguments

data	A matrix, data frame, or tibble.
outliers_only	Logical (default TRUE); if TRUE, only rows actually flagged as outliers are returned.

Value

A tibble with a column identifying which feature each row came from, plus value, .outlier, and .outlier_type.

Examples

```
df <- data.frame(a = c(1, 2, 3, 4, 100), b = c(10, 12, 11, 9, 8))
detect_outliers(df)
detect_outliers(df, outliers_only = FALSE)
```

distribution_plot *Distribution Plot*

Description

Ridgeline density plot of every numeric column in data, optionally normalized/standardized first so columns on different scales stay comparable.

Usage

```
distribution_plot(data, method = "range")
```

Arguments

data	A data frame or tibble.
method	Normalization applied before plotting: "range" (default, via range_norm), "min_max" (via min_max_norm), "z" (via z_norm), or "none"/"raw" to plot the values as-is.

Value

A ggplot object.

Examples

```
df <- data.frame(normal_dist = rnorm(1000, -10, .5), poisson_dist = rpois(1000, 5))
distribution_plot(df)
distribution_plot(df, method = "z")
distribution_plot(df, method = "none")
```

`distribution_test` *Check a Vector or Data Frame's Distribution*

Description

Runs both `is_normal` and `is_poisson` against a numeric vector, or against every numeric column of a data frame.

Usage

```
distribution_test(x)
```

Arguments

`x`	A numeric vector, matrix, or data frame/tibble.

Value

A tibble of test results (with a `feature` column when `x` has multiple columns).

Examples

```
distribution_test(rpois(n = 1000, lambda = 2))
distribution_test(mtcars)
```

`export_interactive_pipeline` *Export the computational graph to a self-contained interactive HTML file for stakeholders*

Description

Export the computational graph to a self-contained interactive HTML file for stakeholders

Usage

```
export_interactive_pipeline(
  all_objects,
  file = here::here("R", "workflow_viz", "full_compute_graph.html"),
  direction = "LR",
  hierarchical = TRUE,
  title = getOption("pipeline.study_name", "Computational Pipeline"),
  subtitle = "Computational Pipeline"
)
```

Arguments

all_objects	A list of FilePath objects or an igraph object.
file	Path to the output HTML file. Default is here::here("R", "workflow_viz", "full_compute_graph.html")
direction	"LR" (default) or "UD".
hierarchical	Logical. Default TRUE.
title	Chart title. Defaults to the ‘pipeline.study_name‘ option, or "Computational Pipeline" if unset.
subtitle	Chart subtitle.

Value

The saved file path (invisibly).

---

export_subgraph *Export a focused interactive subgraph centered around a focal report or stage*

Description

Export a focused interactive subgraph centered around a focal report or stage

Usage

```
export_subgraph(
  all_objects,
  focal_node = NULL,
  stage = NULL,
  order = 1,
  file = NULL
)
```

Arguments

all_objects	List of FilePath objects or an igraph object.
focal_node	Character name of the node to focus on (e.g. "EDA_Report").
stage	Character name of the stage to filter by (e.g. "03_Imputation").
order	Degree of neighborhood around focal_node (default 1).
file	Destination HTML path.

Value

File path invisibly.

extract_all_xlsx_tokens

Extract Every '.xlsx' File Name From Text

Description

Extract Every '.xlsx' File Name From Text

Usage

```
extract_all_xlsx_tokens(x, xlsx_token_re = xlsx_name_re)
```

Arguments

x	Character vector, e.g. code lines.
xlsx_token_re	Regular expression for the file-name part. The default matches a name ending in '.xlsx': right after a quote or path separator it may contain spaces and punctuation other than quotes and separators; elsewhere only letters, digits, '_', '.', and '-'. Names followed by '(' (function calls such as 'read.xlsx()') are skipped.

Value

A list the length of 'x', each element a character vector of the '.xlsx' file names found in that string (empty when none).

See Also

[extract_win_posix_paths()]

Examples

```
extract_all_xlsx_tokens(c("read_excel('out/a.xlsx'); write_xlsx(df, 'b c.xlsx')", "none"))
```

extract_surv_components

Extract Survival Outcome Components and Predictors

Description

Helper utility that molds a survival formula into clean components, through the same **hardhat** formula blueprint used by coxnet (one-hot dummy encoding, no intercept), supporting both 2-parameter `Surv(time, status)` and 3-parameter start/stop `Surv(tstart, tstop, status)` counting process structures.

Usage

```
extract_surv_components(data, outcome, subject_id = NULL)
```

Arguments

data	A data frame.
outcome	A survival formula or Surv expression.
subject_id	Optional subject identifier column name; dropped from ‘data‘ before molding, so it is never treated as a predictor.

Value

A list with elements surv_obj, time, start, status, type, predictors (a numeric tibble, one-hot encoded), pred_names, raw_predictors (the original predictor columns, before encoding, which is what the models’ predict() methods and the explainers take), formula, data (the molded, ‘subject_id‘-free data frame), and blueprint (the hardhat blueprint used, for forge).

---

extract_win_posix_paths

Extract Full ‘.xlsx‘ Paths From Text

Description

Finds the first full Windows or POSIX path ending in an ‘.xlsx‘ file name in each string, and splits it into folder and file name. Separators may repeat (‘C:/data//out‘, or ‘C:\\data‘ as written in R code). Relative paths such as ‘output/t1.xlsx‘ are not full paths and give ‘NA‘.

Usage

```
extract_win_posix_paths(x, xlsx_token_re = xlsx_name_re)
```

Arguments

x	Character vector, e.g. code lines.
xlsx_token_re	Regular expression for the file-name part. The default matches a name ending in ‘.xlsx‘: right after a quote or path separator it may contain spaces and punctuation other than quotes and separators; elsewhere only letters, digits, ‘_‘, ‘.‘, and ‘-‘. Names followed by ‘(‘ (function calls such as ‘read.xlsx()‘) are skipped.

Value

A tibble with one row per element of ‘x‘: ‘dir‘, ‘file‘, and ‘full_path‘ (‘NA‘ where no path was found).

See Also

[extract_all_xlsx_tokens()]

Examples

```
extract_win_posix_paths(c("C:\\data\\table_1.xlsx", "/home/me/out/table-2.xlsx", "none"))
```

extract_yaml_formats *Extract Output Formats From Document YAML Front Matter*

Description

Inspects the YAML header of a Quarto (`.qmd`) or R Markdown (`.Rmd`) file and returns all declared output formats (e.g. `"html"`, `"pdf"`, `"docx"`, `"gfm"`).

Usage

```
extract_yaml_formats(file_path)
```

Arguments

`file_path`	Path to the `.qmd` or `.Rmd` document.

Value

Character vector of lowercase format names, or empty character vector if none found.

features_percent_miss *Calculate Percentage of Missing Data Per Feature*

Description

Computes the count and percentage of missing and complete values for each column in a dataset.

Usage

```
features_percent_miss(data, na_list = NULL)
```

Arguments

`data`	A data frame or tibble.
`na_list`	Optional vector of additional values to treat as missing, passed to `SumNa` (beyond actual `NA`).

Value

A tibble with columns ‘feature‘, ‘SumNa‘, ‘SumComp‘, ‘PctNa‘, ‘PctComp‘ sorted descending by ‘PctNa‘.

Examples

```
df <- data.frame(a = c(1, 2, NA, 4), b = c(NA, NA, 3, 4))
features_percent_miss(df)

df2 <- data.frame(a = c(1, "NA", 3), b = c("", 2, 3))
features_percent_miss(df2, na_list = c("NA", ""))
```

FileOutputs *Construct a FileOutputs object*

Description

User-friendly constructor function for instantiating a `FileOutputs` object.

Usage

```
FileOutputs(
  name,
  path,
  dependencies = list(),
  output = list(),
  renders = TRUE,
  stage = NA_character_,
  artifact_role = "report_source",
  description = NA_character_
)
```

Arguments

`name`	Character. User-supplied human-readable name for the file.
`path`	Character. Absolute or relative file system path.
`dependencies`	List of `FilePath` objects that this file depends on.
`output`	List of `FilePath` objects produced by this file.
`renders`	Logical. Indicates whether the file is a rendering script. Defaults to `TRUE`.
`stage`	Character. Pipeline lifecycle stage.
`artifact_role`	Character. Pipeline role of the artifact. Defaults to `"report_source"`.
`description`	Character. Brief description of the file.

Value

A new `FileOutputs` object.

FileOutputs-class *FileOutputs S4 Class*

Description

Represents an executable or rendering pipeline file that produces downstream output artifacts.

Usage

```
## S4 method for signature 'FileOutputs'
show(object)
```

Arguments

object	A FileOutputs object.

Details

Inherits from FileUses (inheriting upstream dependencies), while adding a list of downstream output FilePath objects produced when this file executes or renders.

Slots

output List of FilePath objects produced by this file.

See Also

FilePath, FileUses, FileOutputs, create_qmd_renderer

FilePath *Construct a FilePath object*

Description

User-friendly constructor function for instantiating a FilePath object.

Usage

```
FilePath(
  name,
  path,
  renders = FALSE,
  stage = NA_character_,
  artifact_role = NA_character_,
  description = NA_character_
)
```

Arguments

name	Character. User-supplied human-readable name for the file.
path	Character. Absolute or relative file system path.
renders	Logical. Indicates whether the file is a rendered file or rendering script. Defaults to FALSE.
stage	Character. Pipeline lifecycle stage.
artifact_role	Character. Pipeline role of the artifact.
description	Character. Brief description of the file.

Value

A new FilePath object.

FilePath-class *FilePath S4 Class*

Description

File paths are unique identifiers for files and are used to track changes in the file system. A file path is a general R, Python, SAS, Quarto, shell, etc. agnostic way to refer to a file. It is not tied to any specific software or programming language, but rather to the file system itself.

Usage

```
## S4 method for signature 'FilePath'
show(object)
```

Arguments

object	A FilePath object.

Details

Represents an individual file node in a computational pipeline dependency graph. A FilePath object encapsulates file-level metadata and state: tracking modification timestamps (mtime), file extensions (file_ext), workflow lifecycle milestones (stage), semantic pipeline roles (artifact_role), and rendering status (renders).

Upon instantiation, the object automatically inspects the target file on disk: if the file exists at path, mtime is populated from file.info()$mtime and file_ext is extracted via tools::file_ext().

Slots

name Character. A user-supplied, human-readable name for the file, used to refer to the file across other parts of the pipeline and dependency graph.

path Character. The absolute or project-relative path to the file on the file system.

mtime POSIXct. The modification time of the file on the file system. Automatically populated from disk during object initialization if the file exists.

file_ext Character. The file extension of the file (e.g., "xlsx", "rds", "qmd"). Automatically populated from the file path during object initialization.

renders Logical. A logical indicating whether the file is a rendered file (i.e. a file that is produced by a rendering process) or a rendering source script. Defaults to FALSE.

stage Character. The workflow lifecycle stage of the file in the pipeline (e.g., "01_EDA", "02_Analysis_Original", "03_Imputation").

artifact_role Character. The semantic role of the file in the pipeline. Standard roles include:

- "raw_data": Original immutable input data (e.g. source spreadsheets).
- "derived_data": Primary processed baseline dataset generated from raw inputs.
- "enhanced_data": Imputed or augmented analytical dataset.
- "intermediate_data": Stage-internal calculation artifact or model.
- "report_source": Executable source document (e.g., Quarto .qmd).
- "deliverable_report": Rendered deliverable (e.g., .pdf, .html).
- "helper_script": Child or modular helper script invoked by parent reports.

- `"reference"`: Lookup tables, crosswalks, or project configuration files.

`description` Character. A brief human-readable description of the file's contents or analytical purpose.

See Also

`FileUses`, `FileOutputs`, `FilePath`, `create_qmd_renderer`

`FileUses` *Construct a FileUses object*

Description

User-friendly constructor function for instantiating a `FileUses` object.

Usage

```
FileUses(
  name,
  path,
  dependencies = list(),
  renders = FALSE,
  stage = NA_character_,
  artifact_role = NA_character_,
  description = NA_character_
)
```

Arguments

`name`	Character. User-supplied human-readable name for the file.
`path`	Character. Absolute or relative file system path.
`dependencies`	List of `FilePath` objects that this file depends on.
`renders`	Logical. Indicates whether the file is a rendered file or rendering script. Defaults to `FALSE`.
`stage`	Character. Pipeline lifecycle stage.
`artifact_role`	Character. Pipeline role of the artifact.
`description`	Character. Brief description of the file.

Value

A new `FileUses` object.

FileUses-class *FileUses S4 Class*

Description

Represents a pipeline file that depends on other upstream files. Extends `FilePath` by adding upstream input dependency tracking.

Usage

```
## S4 method for signature 'FileUses'
show(object)
```

Arguments

`object`	A `FileUses` object.

Details

A `FileUses` node represents a computational artifact whose execution or validity relies upon one or more input `FilePath` objects listed in `dependencies`.

Slots

`dependencies` List of `FilePath` objects that this file depends on.

See Also

`FilePath`, `FileOutputs`, `FileUses`

file_meta_fs *File Metadata as a Tibble*

Description

Vectorized file information from [fs::file_info()], with names and parent folders split out for joining.

Usage

```
file_meta_fs(paths)
```

Arguments

`paths`	Character vector of file paths.

Value

A tibble with one row per path: ‘path‘, ‘file_name‘, ‘dir_name‘ (parent folder name), ‘dir_path‘, ‘m_time‘ (modified), ‘c_time‘ (created, falling back to changed, then modified), ‘size‘ in bytes, and ‘uname‘ (owner, where the platform reports one).

See Also

[scan_data_io()]

Examples

```
f <- tempfile(fileext = ".csv")
writeLines("a,b", f)
file_meta_fs(f)
```

`find_code` *Search for Code Patterns Across a Directory Tree*

Description

Recursively searches R-related files (`.R`, `.Rmd`, `.qmd`, etc.) under `directory` for a string or regex pattern.

Usage

```
find_code(
  directory,
  pattern,
  lines_before = 0,
  lines_after = 0,
  regex = FALSE,
  ignore_case = FALSE,
  include_comments = TRUE,
  exts = c("R", "r", "Rmd", "rmd", "qmd"),
 exclude_dirs = c(".git", "renv", "_freeze", "_book", "_site", ".quarto", "_extensions",
    "cache", ".Rproj.user"),
  return_all_matches = FALSE,
  match_extractor = NULL
)
```

Arguments

`directory`	Character. Root directory to search.
`pattern`	Character. String or regex to search for.
`lines_before, lines_after`	Integers. Context lines around each hit.
`regex`	Logical. If `TRUE`, treat `pattern` as a Perl regex.
`ignore_case`	Logical. Case-insensitive when `TRUE` (works correctly even in fixed-string mode, where base `grep`'s own `ignore.case` argument is silently ignored).
`include_comments`	Logical. If `FALSE`, skip comment-only lines.
`exts`	Character vector of file extensions to include.
`exclude_dirs`	Character vector of directory names to exclude.

`return_all_matches`
: Logical. If `TRUE`, returns one row per extracted match token (via `match_extractor`) instead of one row per matching line.

`match_extractor`
: Optional `function(content_subset)` returning a list of tokens per line; only used when `return_all_matches = TRUE`.

Value

A tibble: `file`, `path`, `line_number`, `line`, and (when `return_all_matches = TRUE`) `match`.

Examples

```
find_code(system.file("R", package = "TempleCBE"), "roxygen2", ignore_case = TRUE)
```

---

find_correlation *Find Highly Correlated Columns*

Description

Identifies numeric columns to drop for multicollinearity, via `findCorrelation` on the pairwise correlation matrix of `data`'s numeric columns. Named `find_correlation()` (not `findCorrelation()`) so it doesn't shadow **caret**'s function of the same name for anyone with both packages loaded.

Usage

```
find_correlation(
  data,
  use = "pairwise.complete.obs",
  method = "pearson",
  cutoff = 0.9,
  verbose = FALSE,
  names = TRUE,
  exact = ncol(data) < 100
)
```

Arguments

`data`	A data frame or tibble.
`use`	Passed to `cor` (default `"pairwise.complete.obs"`).
`method`	Passed to `cor` (default `"pearson"`).
`cutoff`	Absolute correlation above which a column is flagged (default 0.9).
`verbose`	Logical; passed to `findCorrelation`.
`names`	Logical; if `TRUE` (default) return column names instead of indices.
`exact`	Passed to `findCorrelation`; defaults to `ncol(data) < 100` (the original had a bug here referencing an undefined `x` instead of `data` — fixed).

Value

Character vector (or integer indices) of columns to remove.

Examples

```
if (requireNamespace("caret", quietly = TRUE)) {
  find_correlation(mtcars, cutoff = 0.8)
}
```

`find_python` *Locate a Python Interpreter That Can Import a Module*

Description

Search order (first verified hit wins):

1. `getOption("templecbe.python")`
2. `Sys.getenv("TEMPLECBE_PYTHON")`
3. `Sys.getenv("RETICULATE_PYTHON")` and, only when reticulate has already started Python in this session, `reticulate::py_exe()`
4. `Sys.which("python3")`, `Sys.which("python")`
5. Platform-specific well-known locations

Usage

```
find_python(verify = TRUE, module = "pdf2docx")
```

Arguments

`verify`	If `TRUE`, require `import <module>` to succeed.
`module`	Name of the Python module an interpreter must be able to import when `verify = TRUE`: a plain, possibly dotted, module name. Defaults to `"pdf2docx"`.

Details

Each candidate is verified by actually running it, so Windows App Execution Alias stubs (which resolve on PATH but do nothing) are rejected. A probe that takes longer than 60 seconds (running `--version`, or importing `module`) counts as a failure, so one hung interpreter cannot block the search. When `verify = TRUE` the candidate must additionally be able to `import module`. Candidates are probed in a separate process: this function never starts Python inside R, and never starts reticulate's Python, so it is safe to use as a guard before code that would (for example `eval = !is.null(find_python(module = "numpy"))` on a vignette chunk). Install the pinned Python requirements for PDF conversion with `pip install -r` on `system.file("python", "requirements.txt", package = "TempleCBE")`.

The tests that run a real PDF to DOCX conversion (Python, LibreOffice or Word) are opt-in: set the environment variable `TEMPLECBE_RUN_PDF_TESTS=true` to run them. See also `find_sas()` for `TEMPLECBE_RUN_SAS_TESTS`.

Value

Path to a usable interpreter, or `NULL`.

See Also

[check_docx_toolchain()]

`find_sas` *Locate a SAS Executable*

Description

Checks, in order: ‘getOption("templecbe.sas")‘, the ‘SAS_EXE‘ environment variable, the ‘SASROOT‘ environment variable (‘sas.exe‘), ‘sas‘ on the ‘PATH‘, and default SAS install locations (‘SASHome/SASFoundation/<version>‘ under ‘C:/Program Files‘, ‘C:/Program Files (x86)‘, ‘C:‘, ‘D:/Program Files‘, or ‘D:‘ on Windows; under ‘/usr/local/SASHome‘, ‘/opt/sas/SASHome‘, or ‘/opt/SASHome‘ elsewhere), preferring the highest installed version.

Usage

```
find_sas()

sas_available()
```

Details

`sas_available()` is `!is.null(find_sas())`. Guard code that needs SAS with it (for example `eval = TempleCBE::sas_available()` on a vignette chunk), so that code runs only on machines that have SAS.

Value

`find_sas()`: the path to the SAS executable, or `NULL` if none is found. `sas_available()`: `TRUE` or `FALSE`.

Package tests

The package's tests never need SAS: they compare R with reference values that SAS produced once, stored in `tests/testthat/reference/`. A second set of tests runs the real SAS programs and compares their listings with those references. These are opt-in, so that a machine that happens to have SAS does not start slow jobs by accident. Set the environment variable `TEMPLECBE_RUN_SAS_TESTS=true` (and make SAS findable as above) to run them. The same holds for real PDF to DOCX conversions, which need `TEMPLECBE_RUN_PDF_TESTS=true`; see `find_python()`.

See Also

[run_sas_script()], [cbe_sas_macro_path()], [cbe_sas_macro_dir()]

Examples

```
find_sas()
sas_available()
```

find_section_file *Find Section Data File in a Directory*

Description

Find Section Data File in a Directory

Usage

```
find_section_file(data_dir, index)
```

Arguments

data_dir	Directory to search.
index	Section name/token to look for.

Value

File path.

find_soffice *Locate a LibreOffice Headless Binary*

Description

Honours `getOption("templecbe.soffice")` and the `TEMPLECBE_SOFFICE` environment variable before searching the `PATH` and default install locations.

Usage

```
find_soffice()
```

Value

Path to `soffice`/`libreoffice`, or `NULL`.

See Also

[check_docx_toolchain()]

`fit_n_rank` *Fit the Configuration Ranked '.rank' in a Workflow Set*

Description

Ranks the configurations of tuned workflow set results by 'rank_metric', picks the one at position '.rank', and fits it on the full training set with [tune::fit_best()].

Usage

```
fit_n_rank(
  grid_results,
  rank_metric,
  .rank = 1,
  group_wflow = TRUE,
  select_best = FALSE,
  ...
)
```

Arguments

`grid_results`	Tuned workflow set results, the output of [workflowsets::workflow_map()] run with 'control = tune::control_grid(save_workflow = TRUE)', which [tune::fit_best()] needs.
`rank_metric`	Name of the metric to rank by.
`.rank`	Position to fit; 1 (default) is the best.
`group_wflow`	If 'TRUE' (default), rank workflows, each by its best configuration, so '.rank = 2' is the second-best *workflow*. If 'FALSE', rank configurations directly, so '.rank = 2' can be a second configuration of the best workflow.
`select_best`	Passed to [workflowsets::rank_results()]: keep only each workflow's best configuration before ranking.
`...`	Passed to [tune::fit_best()].

Value

A list with 'model', the fitted workflow, and 'tuned_parameters', a one-row tibble of the workflow's 'wflow_id', 'preproc', 'model', tuning parameters, and its [workflowsets::rank_results()] row ('.config', '.metric', 'mean', 'std_err', 'n', 'rank', and 'overall_rank' when 'group_wflow = TRUE').

See Also

[get_model_parameters()]

Examples

```
## Not run:
results <- workflowsets::workflow_map(
  wset, "tune_grid", resamples = folds, grid = 5,
  control = tune::control_grid(save_workflow = TRUE)
```

```
)
best <- fit_n_rank(results, rank_metric = "rmse")
best$tuned_parameters
predict(best$model, new_data)

## End(Not run)
```

`flag_outliers` *Flag and Classify Outliers*

Description

Flag and Classify Outliers

Usage

```
flag_outliers(col)
```

Arguments

`col`	A numeric vector or column.

Value

A one-column-input-turned-tibble with `value`, `.outlier` (factor: `FALSE`, `TRUE`) and `.outlier_type` (factor: `"NONE"`, `"MILD"`, or `"EXTREME"`). A value is flagged only when it lies strictly beyond a fence, so a constant column has no outliers.

Examples

```
flag_outliers(c(1, 2, 3, 4, 5, 100))
```

`fmt_hr` *Format Hazard Ratio and Confidence Interval*

Description

Format Hazard Ratio and Confidence Interval

Usage

```
fmt_hr(hr, conf_low, conf_high, digits = 2)
```

Arguments

`hr`	Point estimate for Hazard Ratio
`conf_low`	Lower 95% confidence limit
`conf_high`	Upper 95% confidence limit
`digits`	Number of decimal places (default: 2)

Value

Formatted string (e.g., "HR 1.25 (95% CI 1.05–1.48)")

fmt_num *Format Numbers to Fixed Decimals*

Description

Format Numbers to Fixed Decimals

Usage

```
fmt_num(x, digits = 2)
```

Arguments

x	Numeric value
digits	Number of decimal places (default: 2)

Value

Formatted character string

fmt_p *Format P-Values*

Description

Formats p-values with institutional threshold (< 0.001).

Usage

```
fmt_p(p)
```

Arguments

p	Numeric p-value

Value

Formatted string (e.g., "p = 0.034" or "p < 0.001")

fmt_pct *Format a Percentage*

Description

Format a Percentage

Usage

```
fmt_pct(x, digits = 1)
```

Arguments

x	Numeric proportion between 0 and 1
digits	Number of decimal places (default: 1)

Value

Formatted character string with '%'

fmt_sig *Format Significant Figures*

Description

Format Significant Figures

Usage

```
fmt_sig(x, digits = 3)
```

Arguments

x	Numeric value
digits	Number of significant figures (default: 3)

Value

Formatted character string

generate_last_names *Procedural Synthetic Last Name Generator*

Description

Generates realistic, pronounceable synthetic surnames using phonetic onsets, vowels, and common surname endings (e.g., '-berg', '-ton', '-man', '-wood', '-ford'). Supports strict exclusion lists (e.g. to guarantee real investigator surnames are never generated) and reproducible seed-based sampling without mutating the global RNG state.

Usage

```
generate_last_names(
  n = 10,
  seed = NULL,
  exclude = character(),
  max_tries = 100L
)
```

Arguments

n	Integer, number of unique names to generate (default 10).
seed	Optional integer, random seed for reproducibility.
exclude	Character vector of surnames (case-insensitive) that must not be generated.
max_tries	Integer, maximum attempts multiplier before throwing an error if distinct names cannot be generated.

Value

A character vector of length 'n' containing synthetic surnames.

Examples

```
# Generate 5 random last names
generate_last_names(5)

# Reproducible generation with exclusions
generate_last_names(3, seed = 1, exclude = c("Madison", "Davison"))
```

generate_pi_names *Generate Random PI Names or Tokens*

Description

Generates random synthetic principal investigator (PI) names, full names, or cryptographic pseudonym tokens ('PI_...').

Usage

```
generate_pi_names(
  n = 1,
  format = c("synthetic", "token", "full_name"),
  prefix = "PI_",
  n_chars = 16,
  rename_fn = NULL,
  seed = NULL,
  exclude = character(),
  allow_collisions = FALSE
)
```

Arguments

n	Integer, number of names to generate (default 1).
format	Character, either ‘"synthetic"‘ (realistic surnames), ‘"full_name"‘ (synthetic first and last name), or ‘"token"‘ (cryptographic hash tokens). Aliases ‘"surname"‘ and ‘"full"‘ are also accepted.
prefix	Character, prefix when format is ‘"token"‘ (default ‘"PI_"‘).
n_chars	Integer, number of hex characters from the hash digest when ‘format = "token"‘ (default 16).
rename_fn	Optional function to rename or format generated tokens/names.
seed	Optional integer, random seed for reproducibility.
exclude	Character vector of surnames or names (case-insensitive) to omit.
allow_collisions	Logical, default FALSE. If TRUE, allows short tokens with n_chars < 6.

Value

A character vector of length ‘n‘ (or a single character string if ‘n = 1‘).

Examples

```
generate_pi_names(1)
generate_pi_names(3, format = "full_name")
generate_pi_names(3, format = "token")
generate_pi_names(3, format = "token", n_chars = 8)
```

---

generate_pseudonym_token

Generate Pseudonym Tokens (Cryptographic Hash)

Description

Generates cryptographic hash pseudonym tokens. When name is supplied, computes deterministic keyed HMAC-SHA256 tokens, which require a secret key (see pi_anonymizer). When name is NULL, generates random tokens (or reproducible, *non-secret* tokens when seed is provided). Supports a custom prefix, a configurable length (e.g. for short tokens), and custom renaming/formatting functions.

Usage

```
generate_pseudonym_token(
  name = NULL,
  n = 1,
  prefix = "PI_",
  key = Sys.getenv("TEMPLECBE_SECRET_KEY", unset = ""),
  n_chars = 16,
  rename_fn = NULL,
  seed = NULL,
  allow_default_key = FALSE,
  allow_collisions = FALSE
)
```

Arguments

name	Optional character vector of names to tokenize. If provided, computes a deterministic HMAC-SHA256 token for each non-NA name; this needs a secret key.
n	Integer, number of tokens to generate when name is NULL (default 1).
prefix	Character string, prefix prepended to each token (default "PI_").
key	Character string, the secret key for HMAC tokenization, used only when name is supplied. Defaults to the environment variable TEMPLECBE_SECRET_KEY. An empty key counts as no key: the call stops unless allow_default_key = TRUE. Prefer the environment variable over writing the key into a script.
n_chars	Integer, number of hex characters from the hash digest to include (default 16). Can be shortened (e.g. 4 or 8) to produce compact tokens, at the cost of more collisions between different names.
rename_fn	Optional function to transform or rename tokens (e.g., tolower, toupper, or a custom function such as function(tok) paste0("INV-", substr(tok, 4, 8))).
seed	Optional integer, random seed for reproducible random token generation when name is NULL. Seeded tokens are derived from the seed alone, so anyone who knows it can regenerate them; do not use them where secrecy matters.
allow_default_key	Logical, default FALSE. If TRUE and no key is available, use the package's public built-in key instead of stopping, and warn that the tokens are NOT secret. This only reproduces output made without a key by earlier versions; do not use it for data you intend to protect.
allow_collisions	Logical, default FALSE. If TRUE, suppresses errors when n_chars < 6 or when distinct input names map to identical pseudonym tokens.

Details

Tokens replace investigator names only; nothing else in a dataset is de-identified (see the scope notes in pi_anonymizer).

Value

Character vector of pseudonym tokens.

See Also

pi_anonymizer for scope and key handling.

Examples

```
# Random tokens need no key
generate_pseudonym_token(n = 3)

# Short tokens (8 hex chars)
generate_pseudonym_token(n = 3, n_chars = 8)

# Custom renaming function
generate_pseudonym_token(n = 2, rename_fn = tolower)

# Deterministic tokens need a secret key. In real use keep it in the
# TEMPLECBE_SECRET_KEY environment variable (see ?pi_anonymizer), never in a
# script you commit. Here a throwaway value is set for this example only.
if (requireNamespace("withr", quietly = TRUE)) {
  withr::with_envvar(c(TEMPLECBE_SECRET_KEY = "example-only-not-a-real-key"), {
    print(generate_pseudonym_token("Franklin"))
    print(generate_pseudonym_token(c("Smith", "Jones")))
  })
}
```

---

get_dataset_info *Summarize a Data Frame or Joint Model's Columns and Components*

Description

Per-column metadata: class, variable label (if set via **labelled**), mean/sd for numeric columns, most-frequent value, distinct-value count, and missingness. survival::Surv columns are summarized from their underlying time/status or start/stop counting-process matrix rather than unrolled as plain numerics. When provided a joint_model object, summarizes the fitted training data and attaches model metadata. When provided a list of data frames (an R database), returns a consolidated data dictionary across all tables.

Usage

```
get_dataset_info(x, ...)

## S3 method for class 'data.frame'
get_dataset_info(
  x,
  subject_id = NULL,
  dataset_name = NULL,
  database_name = NULL,
  dataset_label = NULL,
  ...
)

## S3 method for class 'list'
get_dataset_info(x, subject_id = NULL, database_name = NULL, ...)
```

```
## S3 method for class 'joint_model'
get_dataset_info(x, ...)

proc_contents(x, ...)
```

Arguments

x	A data frame, tibble, named list of data frames, or a fitted joint_model object.
...	Additional arguments passed to methods.
subject_id	Optional character string specifying the subject identifier column for repeated-measures longitudinal datasets to audit time-varying vs. baseline features.
dataset_name	Optional character string overriding the displayed dataset name.
database_name	Optional character string specifying the parent database name.
dataset_label	Optional character string overriding the dataset-level description.

Value

A tibble with one row per column: dataset_name, (optional database_name and dataset_label), labels, columns, class, mean, sd, most_freq, n_distinct, SumNa, PctNa, and optionally variable_type when subject_id is specified.

Examples

```
get_dataset_info(mtcars)
```

---

get_model_parameters *Tuning Parameters of a Ranked Workflow in a Workflow Set*

Description

Ranks the workflows of tuned workflow set results by ‘rank_metric‘, each workflow represented by its best configuration, and returns the preprocessor, model, and best tuning parameters of the workflow at position ‘.rank‘.

Usage

```
get_model_parameters(workflowset_results, rank_metric, .rank = 1)
```

Arguments

workflowset_results	Tuned workflow set results, the output of [workflowsets::workflow_map()].
rank_metric	Name of the metric to rank by, e.g. ‘"rmse"‘ or ‘"brier_survival_integrated"‘.
.rank	Position to return; 1 (default) is the best workflow.

Value

A one-row tibble with ‘wflow_id‘, ‘preproc‘, ‘model‘, and one column per tuning parameter.

See Also

[fit_n_rank()] to also fit the chosen configuration.

Examples

```
## Not run:
results <- workflowsets::workflow_map(wset, "tune_grid", resamples = folds, grid = 5)
get_model_parameters(results, "rmse")
get_model_parameters(results, "rmse", .rank = 2)

## End(Not run)
```

`get_render_plan` *Get the topologically sorted list of QMDs that need to be re-rendered*

Description

Get the topologically sorted list of QMDs that need to be re-rendered

Usage

```
get_render_plan(all_objects)
```

Arguments

`all_objects`	The full list of FilePath, FileUses, and FileOutputs objects.

Details

The graph's identity is the node `name`, not the file `path`: a file declared under two names is two nodes, so a report that reads it under one name is not planned when the report that writes it under the other is. This is reported with a warning, as is one name used for two files, and so are two producers that declare the same output file. Two objects of `all_objects` with the same name are an error. See `as_igraph`.

Value

A list of FileOutputs objects (the QMDs) in the correct render order.

glmnet_IBS *Integrated Brier Score of a Penalized Cox Model on Start/Stop Survival Data*

Description

Tunes an elastic-net Cox model on the analysis set of one resampling split and scores it on the assessment set by the integrated Brier score (IBS). Lower is better. A convenience wrapper around [cv_coxnet()] for data in counting-process (start/stop) layout, kept for existing analysis code; new code can call [cv_coxnet()] and [nested_cv_coxnet()] directly.

Usage

```
glmnet_IBS(
  object,
  alpha = 1,
  recipe,
  feature_names,
  time_data = NULL,
  formula = NULL,
  internal_folds = 5,
  id_col = "id",
  start_col = "tstart",
  stop_col = "tstop",
  status_col = "status",
  censoring_weights = c("ipcw", "none"),
  prep_on = c("baseline", "all"),
  eval_time = NULL,
  metric = "brier_survival_integrated",
  rule = c("min", "1se"),
  covariates = c("path", "baseline"),
  failure_ibs = NA_real_,
  ...
)
```

Arguments

object	An ‘rsplit‘ (e.g. one element of ‘rsample‘ resamples).
alpha	Elastic net mixing parameter: 1 is lasso, 0 is ridge.
recipe	An unprepped [recipes::recipe()].
feature_names	Character vector of baked predictor column names, or a function that takes the baked analysis set and returns them – for recipes whose output columns vary by fold, such as ‘step_pca(threshold =)‘.
time_data	Optional data frame with columns ‘start_col‘ and ‘stop_col‘; its positive stop times are used as ‘eval_time‘ when that is ‘NULL‘.
formula	Optional character string of ‘+‘-separated feature names (or a one-sided formula) to restrict the model to a subset of ‘feature_names‘. Defaults to all of them.
internal_folds	Number of subject-grouped folds used to choose the penalty.

`id_col`, `start_col`, `stop_col`, `status_col`
: Column names of the subject identifier, interval start, interval stop, and event indicator (1 = event).

`censoring_weights`
: Only ‘"ipcw"‘ is supported. ‘"none"‘, which scored interval rows without adjusting for censoring, was removed in TempleCBE 0.3.0; install TempleCBE 0.2.0 to reproduce results that used it.

`prep_on`
: ‘"baseline"‘ (default) preps ‘recipe‘ on each subject’s first interval in the analysis set; ‘"all"‘ preps on every analysis row.

`eval_time`
: Evaluation times for the IBS. Defaults to the stop times of ‘time_data‘, or else deciles of the analysis-set event times.

`metric`
: Name of the yardstick survival metric that chooses the penalty, e.g. ‘"brier_survival_integrated"‘ or ‘"concordance_survival"‘.

`rule`
: ‘"min"‘ or ‘"1se"‘: report the penalty at ‘lambda.min‘ or ‘lambda.1se‘.

`covariates`
: ‘"path"‘ or ‘"baseline"‘; see [cv_coxnet()].

`failure_ibs`
: IBS reported when the model cannot be fit (default ‘NA‘); a warning gives the reason.

`...`
: Further arguments passed to [glmnet::glmnet()], such as ‘cox.ties‘ (‘"breslow"‘ or ‘"efron"‘). glmnet 5.0 defaults to Breslow and 5.1 to Efron, so pass ‘cox.ties‘ to keep results stable across glmnet versions.

Details

1. **Preprocessing.** ‘recipe‘ is prepped on the analysis set – only each subject’s first interval when ‘prep_on = "baseline"‘ – and baked onto both sets. The recipe must keep ‘id_col‘, ‘start_col‘, ‘stop_col‘, and ‘status_col‘ in its output (e.g. with an ‘"id variable"‘ role). 2. **Tuning.** [cv_coxnet()] chooses the penalty on the baked analysis set with ‘internal_folds‘ folds grouped by subject, by ‘metric‘ (the IBS by default). Earlier versions used [glmnet::cv.glmnet()], which chose the penalty by concordance on folds of rows, splitting subjects’ intervals between folds. 3. **Scoring.** The model, refit on the whole analysis set, predicts each assessment-set subject’s survival from the analysis-set Breslow baseline hazard along their covariate path (see [cv_coxnet()]), and is scored by [yardstick::brier_survival_integrated()] with inverse-probability-of- censoring (Graf) weights estimated on the analysis set.

Value

A tibble with columns ‘IBS‘, ‘lambda‘, ‘term‘, ‘estimate‘, and ‘alpha‘: one row per feature (including those the penalty set to 0). When the fit fails, a single row with ‘IBS = failure_ibs‘, ‘lambda = NA‘, and ‘alpha‘.

References

Graf E, Schmoor C, Sauerbrei W, Schumacher M (1999). Assessment and comparison of prognostic classification schemes for survival data. *Statistics in Medicine*, 18(17-18), 2529-2545.

See Also

[cv_coxnet()], [nested_cv_coxnet()], [tune_over_alpha()], [summarize_tune_results()]

Examples

```
if (requireNamespace("glmnet", quietly = TRUE) &&
    requireNamespace("yardstick", quietly = TRUE) &&
    requireNamespace("rsample", quietly = TRUE)) {
  set.seed(1)
  # Synthetic start/stop data: 60 subjects, up to 3 intervals each
  long <- do.call(rbind, lapply(1:60, function(i) {
    k <- sample(1:3, 1)
    stops <- c(5, 10, 20)[1:k]
    risk <- rnorm(1)
    data.frame(subject_id = i, tstart = c(0, head(stops, -1)), tstop = stops,
               status = c(rep(0, k - 1), rbinom(1, 1, plogis(risk))),
               x1 = risk + rnorm(k, sd = 0.2), x2 = rnorm(k))
  }))
  rec <- recipes::recipe(~ ., data = long) |>
    recipes::update_role(subject_id, tstart, tstop, status, new_role = "id variable") |>
    recipes::step_range(recipes::all_numeric_predictors())
  split <- rsample::group_initial_split(long, group = subject_id)
  glmnet_IBS(split, alpha = 0.5, recipe = rec, feature_names = c("x1", "x2"),
             time_data = unique(long[, c("tstart", "tstop")]), internal_folds = 3,
             id_col = "subject_id", cox.ties = "breslow")
}
```

`graf_weights` *Inverse Probability of Censoring Weights (Graf et al.)*

Description

Computes the weights time-dependent survival metrics need so that subjects lost to follow-up don't bias the score: at evaluation time t, a subject still under observation ($T > t$) gets $1/G(t)$; a subject with an event by t gets $1/G(T^-)$; a subject censored by t gets 0, as their status at t is unknown.

Usage

```
graf_weights(truth, eval_time, censoring, trunc = 0.05)
```

Arguments

`truth`	A right-censored [survival::Surv()] object, one element per subject (see [surv_subject_truth()]).
`eval_time`	Numeric vector of evaluation times.
`censoring`	A [censoring_km()] object, estimated on the training data.
`trunc`	Lower bound for G, capping weights at '1 / trunc' (default 0.05, as in tidymodels' 'parsnip').

Details

The weights assume follow-up starts at time 0 for every subject: they do not correct for delayed entry (left truncation).

Value

A numeric matrix with one row per element of 'truth' and one column per 'eval_time'.

References

Graf E, Schmoor C, Sauerbrei W, Schumacher M (1999). Assessment and comparison of prognostic classification schemes for survival data. *Statistics in Medicine*, 18(17-18), 2529-2545.

See Also

[add_graf_weights()], [censoring_km()]

Examples

```
if (requireNamespace("survival", quietly = TRUE)) {
  train <- survival::Surv(c(2, 4, 6, 8), c(0, 1, 0, 1))
  test <- survival::Surv(c(5, 7), c(1, 0))
  graf_weights(test, eval_time = c(3, 6), censoring = censoring_km(train))
}
```

---

graph_setops *Intersect, Subtract, or Union Two Graph Objects*

Description

Set operations across two graph objects, regardless of their concrete representation (`igraph`/`tbl_graph`, or a **visNetwork** htmlwidget such as `funviewR::plot_dependency_graph()` returns). Both graphs are normalized via `as_database`, combined with the `database_setops` verbs (`dplyr::intersect()`/`dplyr::setdiff()`/`dplyr::union()` on the `cbe_database` class), and rebuilt into an `igraph` with `database_to_igraph`.

Usage

```
graph_intersect(x, y, directed = TRUE)

graph_subtract(x, y, directed = TRUE)

graph_union(x, y, directed = TRUE)
```

Arguments

`x`, `y`	Graph objects: `igraph`/`tbl_graph` objects, or **visNetwork** htmlwidgets.
`directed`	Logical, whether the rebuilt graph is directed (default `TRUE`).

Details

These stay dedicated functions rather than `intersect.igraph()`/`union.igraph()`/`setdiff.igraph()` methods because **igraph** already registers its own `union.igraph()` on the same generic – S3 method tables are global, so defining one here would silently collide with igraph's. Going through `as_database` first sidesteps that entirely: dispatch happens on the `cbe_database` class this package owns, not on `igraph`.

`graph_intersect()` keeps nodes/edges present in *both* `x` and `y`, taking attribute values from `x` only. `graph_subtract()` keeps nodes/edges present in `x` but *not* in `y` (edges are additionally constrained to only reference surviving nodes, so the result is always well-formed). `graph_union()` combines both graphs' nodes and edges, coalescing attribute values so neither side's data is lost on overlap

– the same union-and-coalesce behavior as join_pipelines, generalized to any graph type via as_database rather than tbl_graph specifically.

All three are set operations – which nodes/edges exist where – not a value-level comparison of attributes on nodes/edges that happen to match. For that, run cbe_compare_df on the two as_database results directly (or on the two graphs themselves).

Value

An igraph object. Call as_database on the result to get the underlying node/edge tibbles.

Examples

```
g1 <- igraph::graph_from_data_frame(
  data.frame(from = c("a", "b"), to = c("b", "c")),
  vertices = data.frame(name = c("a", "b", "c"))
)
g2 <- igraph::graph_from_data_frame(
  data.frame(from = "a", to = "b"),
  vertices = data.frame(name = c("a", "b"))
)
graph_intersect(g1, g2)
graph_subtract(g1, g2)
graph_union(g1, g2)
```

---

guard_public_remote *Guard a Public Remote Against Accidental Direct Push*

Description

Records the URL(s) of the public remote in git config (cleanpublish.publicurl) and makes sure the pre-push hook is installed. The hook refuses a git push whose push URL is one of them, after git has applied its own rewriting (url.<base>.insteadOf, pushInsteadOf) and the URLs have been normalised: case, a trailing .git or slash, a scheme, a user name, host:path and backslash notation do not matter, and a local directory counts by its physical path as well. Nothing else is resolved (a symbolic link on a remote host, a changed DNS name), and a URL of another spelling is not the same URL.

Usage

```
guard_public_remote(public_url, repo_root = NULL, allow_unmatched = FALSE)
```

Arguments

public_url	The name of a configured remote (its fetch and push URLs, as git remote get-url --all and --push --all report them, are stored), or a URL. A URL must match the push URL of at least one configured remote after normalising, so that a fence that guards nothing is an error, unless allow_unmatched = TRUE. It must not contain whitespace, which the hook cannot compare word by word.
repo_root	Path to the git repository root. Defaults to the git toplevel of the working directory (an error when that is not in a repository).
allow_unmatched	Logical (default FALSE). Set TRUE to fence a URL that no configured remote has yet, for a remote that will be added later.

Details

The one push that goes through is the one `clean_publish()` makes: it sets the environment variable `TEMPLECBE_PUBLISHING=1`, and with that variable the hook lets through the clean commits `clean_publish()` recorded, and nothing else. The variable bypasses the fence only: any other ref, the private branch for one, is still refused.

After writing, the function checks the result. It runs the hook by hand, as git does, for every configured remote and for every fenced URL, with `TEMPLECBE_PUBLISHING` unset, and stops with an error when the hook is missing, is not executable, or does not refuse a push to a remote the fence matches: a fence that is not confirmed is not left to guard nothing in silence.

What the guard does not stop. It is a hook of command-line `git`. It does not stop `git push --no-verify`, a push made by a client that does not run hooks (the **gert** package, which **usethis** uses, is one), a push after `core.hooksPath` has been pointed elsewhere or the hook has been deleted or made non-executable, and a push to the URL in a spelling that is not one of the fenced ones, for example an explicit URL that names the same repository through another host name. It checks pushes, not reads. The safest procedure is to publish from a throwaway clone and to keep no pushable public remote in the clone you work in.

Value

The normalized fenced URL(s), invisibly.

Examples

```
## Not run:
# the remote of the public repository, by name ...
guard_public_remote("public")
# ... or by URL, which must be the push URL of a configured remote
guard_public_remote("https://github.com/OWNER/NEW-PUBLIC-REPO.git")
# a URL for a remote that is added after this call
guard_public_remote("https://github.com/OWNER/NEW-PUBLIC-REPO.git", allow_unmatched = TRUE)

## End(Not run)
```

`infix_helpers` *Pattern Matching and Logical-Negation Infix Operators*

Description

A small family of `grepl()`/`%in%` wrappers as base functions and matching infix operators.

Usage

```
like(vector, pattern, ignore.case = FALSE, fixed = FALSE, perl = FALSE)

vector %like% pattern

ilike(vector, pattern)

vector %ilike% pattern

flike(vector, pattern)
```

```
vector %flike% pattern

plike(vector, pattern)

vector %plike% pattern

notin(x, table)

x %!in% table

x %notin% table
```

Arguments

pattern	Pattern to match (for the like family).
ignore.case, fixed, perl	Logical flags controlling how pattern is matched, as in grepl.
x, vector	A vector to test.
table	Vector of values to test against (for notin()/%notin%/%!in%).

Value

A logical vector.

---

install.packages.no_lock

Install a Package, Bypassing an Existing Lock

Description

Install a Package, Bypassing an Existing Lock

Usage

```
install.packages.no_lock(packages)
```

Arguments

packages	Character vector of package name(s) to install.

Value

Invisibly, NULL (as per install.packages).

Examples

```
## Not run:
install.packages.no_lock("dplyr")

## End(Not run)
```

is.int — *Is a Vector Composed of Integer-Valued Numbers*

Description

Is a Vector Composed of Integer-Valued Numbers

Usage

```
is.int(col)
```

Arguments

col	A numeric vector.

Value

A single logical.

Examples

```
is.int(sample(-100:100, size = 500, replace = TRUE))
is.int(runif(500))
```

is_normal — *Test Whether a Vector Looks Normally Distributed*

Description

Runs two normality tests on the finite values of ‘col‘:

Usage

```
is_normal(col, alpha = 0.1)
```

Arguments

col	A numeric vector. ‘NA‘, ‘NaN‘, and infinite values are dropped.
alpha	Significance level for ‘distribution.test‘.

Details

* **Shapiro-Wilk** ([stats::shapiro.test()]), for 3 to 5000 values. It is left out above 5000 rather than run on a random subsample, so results never depend on the random number generator. * **Lilliefors** ([nortest::lillie.test()]), for 5 or more values: the Kolmogorov-Smirnov test corrected for estimating the mean and standard deviation from the same data. A plain KS test against ‘pnorm(mean(x), sd(x))‘ gives p-values that are far too large.

Both tests are deterministic. In large samples they reject normality for departures too small to matter, so read them alongside a plot such as [distribution_plot()] or a normal Q-Q plot.

Value

A tibble with one row per test run: ‘statistic‘, ‘p.value‘, ‘method‘, ‘distribution.test‘ (‘TRUE‘ when ‘p.value >= alpha‘, i.e. normality is not rejected), ‘p_value_sig‘, and ‘distribution‘. It has no rows when neither test applies (fewer than 3 values, or all values equal).

References

Lilliefors HW (1967). On the Kolmogorov-Smirnov test for normality with mean and variance unknown. *Journal of the American Statistical Association*, 62(318), 399-402.

Examples

```
set.seed(1)
is_normal(rnorm(1000, mean = 5, sd = 3))
is_normal(runif(1000, min = 2, max = 4))
```

---

is_poisson *Test Whether a Vector Looks Poisson-Distributed*

Description

Runs a chi-squared goodness-of-fit test: observed counts (binned at roughly equal-probability quantiles of the fitted Poisson distribution, so expected counts stay adequate regardless of where the data sits) vs. Poisson-expected counts, with one degree of freedom subtracted for the estimated rate. The test is deterministic — no simulated comparison sample is drawn.

Usage

```
is_poisson(col)
```

Arguments

`col`	A numeric vector.

Details

A Kolmogorov-Smirnov test is deliberately *not* used here: the KS statistic's null distribution assumes a continuous CDF, and the Poisson distribution is discrete with real point masses, which inflates the KS statistic (and deflates its p-value) regardless of true fit — the chi-squared test is the standard, correctly-calibrated tool for discrete/count goodness-of-fit. `is_normal` uses the Lilliefors variant of the Kolmogorov-Smirnov test, which suits the continuous normal distribution.

Since the Poisson distribution's support is the non-negative integers, this returns an empty tibble for vectors containing negative values or fewer than 2 observations.

Value

A one-row tibble of test results (empty if `col` isn't valid count data, e.g. it has negative values).

Examples

```
is_poisson(rpois(n = 1000, lambda = 2))
is_poisson(runif(1000, min = 2, max = 4))
```

`joint_model` *Joint Survival-Status-Time Model*

Description

Fits a coordinated trio of predictive models on clinical survival data:

1. **Survival Model** (`coxnet`): Penalized Cox proportional hazards model on `Surv(time, status)` or start/stop counting-process data `Surv(tstart, tstop, status)` via `coxnet` and `cv_coxnet`, correctly handling right-censoring and time-varying intervals.
2. **Status Model** (`status`): Binary event classification on `status ~ x` via penalized logistic regression (**glmnet**) or bagged trees (**baguette**), with optional probability calibration via **probably**.
3. **Time Model** (`time`): Continuous duration regression on `time ~ x` via penalized linear regression (**glmnet**) or bagged trees (**baguette**).

Usage

```
joint_model(
  data,
  outcome = survival::Surv(time, status) ~ .,
  subject_id = NULL,
  engine = c("glmnet", "baguette", "stacks"),
  calibration = TRUE,
  mixture = 1,
  penalty = NULL,
  eval_time = NULL,
  covariates = c("path", "baseline"),
  ...
)
```

Arguments

`data`	A data frame containing the survival outcome and predictors.
`outcome`	A formula containing a `Surv` outcome, such as `Surv(time, status) ~ .` or `Surv(tstart, tstop, status) ~ .`.
`subject_id`	Optional character string specifying the subject identifier column for counting-process (start/stop) data.
`engine`	Character string specifying the modeling engine: `"glmnet"` (default), `"baguette"` (bagged decision trees), or `"stacks"` (the `"baguette"` models plus a Cox meta-learner whose predictions `predict()` adds as extra columns; see Details).
`calibration`	Logical; whether to fit a probability calibration model on the status predictions using **probably** (default `TRUE`), fitted on out-of-fold predictions; see Details.
`mixture`	Elastic net mixing parameter for **glmnet** models (default 1 for lasso).
`penalty`	Penalty value for the `coxnet` survival model. If `NULL` (default), tuned automatically via internal cross-validation using the Integrated Brier Score. The **glmnet** status and time models always choose their own penalty (`lambda.min`).
`eval_time`	Optional vector of evaluation times for dynamic survival probabilities. Defaults to deciles of uncensored event times.

covariates	"path" (the default) or "baseline"; as in cv_coxnet. Used only when penalty is NULL, to score the internal cross-validation that tunes the penalty of the coxnet model on start/stop data.
...	Additional arguments passed to cv_coxnet (when penalty is NULL) or coxnet, and so to glmnet for the survival model only.

Details

With engine = "stacks" the status and time models are the bagged trees of engine = "baguette", and a penalized Cox meta-learner (**glmnet**) is also fit on the standardized predictions of the three models and stored as stack_model. predict.joint_model returns the meta-learner's linear predictor and risk score as the extra columns .pred_stack_linear_pred and .pred_stack_risk_score; the survival, status and time predictions are exactly those of "baguette", and the **stacks** package is not involved. A warning says so.

This joint framework enables clinical researchers to investigate and contrast true survival modeling against naive classification and regression proxies as discussed in clinical literature (e.g., Rizopoulos 2015, PMC4503792).

Predictors are processed through the **hardhat** coxnet formula blueprint (one-hot dummy encoding, no intercept), the same preprocessing used by coxnet and cv_coxnet, so the status and time sub-models see exactly the columns the survival sub-model does, and 'predict()' re-applies the training encoding via forge rather than recomputing dummy columns from scratch.

The status calibrator is fitted on out-of-fold predictions, never on the training rows' own predictions, which are overfit (a bagged tree classifies its training rows almost perfectly, and a calibrator fitted on them amplifies the overfit). For engine = "glmnet" these are cv.glmnet's prevalidated predictions at lambda.min; for the bagged trees, the predictions of an inner 5-fold cross-validation. With start/stop data and a subject_id, the folds are grouped by subject, so a subject's intervals are never split between the model and the predictions used to calibrate it; the same folds select the penalty of the glmnet status and time models. When out-of-fold predictions cannot be produced, a warning says so and the status probabilities are returned uncalibrated.

Value

An S3 object of class c("joint_model", "hardhat_model") with elements:

coxnet_model	Fitted penalized Cox proportional hazards model.
status_model	Fitted binary event classification model.
time_model	Fitted continuous follow-up duration model.
stack_model	Fitted Cox meta-learner (if engine = "stacks"); predict() returns its output as .pred_stack_linear_pred and .pred_stack_risk_score.
calibration_model	Probability calibration model from **probably** (NULL if calibration = FALSE or if it was skipped with a warning).
components	List of extracted outcome variables, formulas, the hardhat blueprint, and the raw (pre-blueprint) predictor columns, raw_predictors.
engine	Selected modeling engine.
eval_time	Evaluation horizons used for survival scoring.

See Also

cv_joint_model, nested_cv_joint_model, cv_coxnet, glmnet_IBS

Examples

```
if (requireNamespace("glmnet", quietly = TRUE) &&
    requireNamespace("survival", quietly = TRUE)) {
  set.seed(42)
  df <- data.frame(
    time = stats::rexp(60, rate = 0.05),
    status = stats::rbinom(60, 1, 0.6),
    x1 = stats::rnorm(60),
    x2 = stats::rnorm(60)
  )
  fit <- joint_model(df, survival::Surv(time, status) ~ x1 + x2)
  print(fit)
  preds <- predict(fit, new_data = df[1:5, ])
}
```

---

join_pipelines *Join two computational pipeline graphs*

Description

Relational union of two pipeline subgraphs using tidygraph::graph_join(by = "name"). Nodes with identical names are coalesced, and their directed dependency and output edges are combined without duplicating nodes.

Usage

```
join_pipelines(graph_a, graph_b, ...)
```

Arguments

graph_a	A list of FilePath objects, igraph, or tbl_graph.
graph_b	A list of FilePath objects, igraph, or tbl_graph.
...	Additional arguments passed to tidygraph::graph_join.

Details

For a node that is in both graphs, a placeholder counts as missing: "Other" as a stage, "unspecified" as an artifact_role, an empty string, and NA. A graph that merely reads a file (a bare copy of it in a dependencies list) has only placeholders for its stage, role and description, and must not hide what the graph that produces it says, so the real value wins whichever graph is given first. When both graphs hold a real value, the first graph's is kept. The node is stale (and renders) if it is in either graph; a node that is stale in only one graph is drawn as stale, with the stale colour and tooltip. The colour, shape and tooltip of a node otherwise come from the graph that holds more real details about it. An edge that both graphs hold (same from, to and type) is one edge.

Nodes are matched by name, as everywhere in the pipeline graph (see as_igraph).

Value

A merged tbl_graph object.

keep_only *Keep Only Specified Objects in an Environment*

Description

Removes every object in the caller's environment except the ones named in `vector`. Prompts for confirmation in interactive sessions unless `.dontask = TRUE`; proceeds without prompting in non-interactive sessions (scripts, `R CMD check`, knitr rendering), since `readline()` would otherwise hang there.

Usage

```
keep_only(vector, .dontask = FALSE)
```

Arguments

`vector`	Character vector of object names to keep.
`.dontask`	Logical (default `FALSE`); skip the confirmation prompt.

Details

`vector` must be a character vector of object names, so the names are quoted: `keep_only(c("a", "b"))`, not `keep_only(c(a, b))`. Every name has to be an object of the calling environment itself (not of one above it); otherwise the function stops before it removes anything. A misspelt name used to pass unnoticed and remove every object, the one that was meant to be kept included.

Called from inside a function, `keep_only()` works on that function's own environment, so the function's arguments are removed too unless they are named in `vector`.

Value

Invisibly, `NULL`.

Examples

```
e <- new.env()
local({a <- 1; b <- 2; keep_only("a", .dontask = TRUE)}, envir = e)
ls(e)
```

---

km_summary_to_prism *Convert a Kaplan-Meier Summary Table to a GraphPad Prism Survival Table*

Description

Expands a summary-by-time table of event and censoring counts (as from 'summary(survival::survfit(...))') into Prism's survival data layout: one row per subject, an 'X' column of times, and one column per group holding 1 for an event, 0 for censored, and 'NA' for the other groups.

Usage

```
km_summary_to_prism(
  x,
  sheet = NULL,
  time_col = "time",
  strata_col = "strata",
  n_event_col = "n.event",
  n_censor_col = "n.censor",
  strata_levels = NULL,
  drop_time0 = TRUE,
  validate_totals = TRUE,
  out_xlsx = NULL,
  out_xlsx_paste = NULL
)
```

Arguments

x	A data frame, or a path to an '.xlsx', '.xls', '.csv', or '.txt' file, with one row per time per group.
sheet	Sheet name or index when 'x' is an Excel file.
time_col, strata_col, n_event_col, n_censor_col	Column names of the time, group, number of events, and number censored.
strata_levels	Optional group order for the output columns. Every group in the data must be listed.
drop_time0	Drop rows at time 0 (default 'TRUE').
validate_totals	If 'TRUE' (default) and 'x' has an 'n.risk' column, warn when a group's 'n.risk' at its earliest time differs from its total events plus censored.
out_xlsx	Optional path to write the table as Excel (requires 'writexl').
out_xlsx_paste	Optional path to write a copy with blanks instead of 'NA', convenient for pasting into Prism.

Value

A data frame with column 'X' and one integer column per group, with a '"notes"' attribute describing the layout.

Examples

```
if (requireNamespace("survival", quietly = TRUE)) {
  fit <- survival::survfit(survival::Surv(time, status) ~ sex, data = survival::lung)
  s <- summary(fit, censored = TRUE)
  km <- data.frame(time = s$time, strata = s$strata, n.risk = s$n.risk,
                   n.event = s$n.event, n.censor = s$n.censor)
  head(km_summary_to_prism(km))
}
```

list_stage_data_artifacts *Inspect and categorize all on-disk data artifacts for a pipeline stage*

Description

Scans a directory for data files (.rds, .xlsx, .csv) and distinguishes primary pipeline handoff datasets from intermediate analysis tables.

Usage

```
list_stage_data_artifacts(
  stage_dir,
  stage = NA_character_,
  primary_patterns = c("^DATA\\.rds$", "^INDEX\\.rds$", "Index_Scores",
    "imputed_DATA")
)
```

Arguments

stage_dir	Directory path to scan (e.g. analysis/stage_a/data).
stage	Optional stage identifier to tag the artifacts with.
primary_patterns	Regex pattern identifying primary handoff datasets.

Value

A tibble of data artifacts with name, path, stage, role, size_kb, and mtime.

make_excel_names *Generate Excel-Compatible Column Names*

Description

Formats column names into sanitized, human-readable strings suitable for Excel output headers.

Usage

```
make_excel_names(names)
```

Arguments

names	A character vector of column names.

Value

Sanitized character vector.

Examples

```
make_excel_names(c("patient_id", "body_mass_index"))
```

manhattan_plot *Manhattan Plot*

Description

Manhattan Plot

Usage

```
manhattan_plot(.data, var, log_p, alpha = 0.05, highlight_significant = TRUE)
```

Arguments

.data	A data frame, e.g. the output of multiple_t_test.
var	Unquoted column of variable labels.
log_p	Unquoted column of -log10(p.value).
alpha	Significance threshold for the reference line/shading (default 0.05).
highlight_significant	Logical (default TRUE); shade the region above the p < alpha line.

Value

A ggplot object.

Examples

```
mtcars |>
  dplyr::mutate(am = factor(am)) |>
  multiple_t_test(.class = "am") |>
  manhattan_plot(var = var, log_p = log_p)
```

min_max_norm *Min-Max Data Normalization*

Description

Normalizes numerical values into [0, 1]. For a data frame, matrix, or tibble, each numeric column is normalized independently against its own min/max, and **non-numeric columns are dropped** — matching the original implementation this was migrated from.

Usage

```
min_max_norm(obj)
```

Arguments

obj	A numeric vector, matrix, data frame, or tibble.

Details

Missing values are preserved and ignored when finding the min/max; only finite values define the range. A constant input (zero range) maps every finite value to `0` instead of `NaN`, and an input with no finite values comes back as all `NA`.

Value

A normalized object: a numeric vector stays a vector; a matrix stays a matrix; a data frame or tibble is reduced to just its (now normalized) numeric columns.

Examples

```
min_max_norm(c(10, 20, 30, 40, 50))
min_max_norm(data.frame(x = c(1, 2, 3), y = c(-10, 0, 10), id = c("a", "b", "c")))
```

`missforest_impute_by_mtry`

Assemble Imputed Columns From Their Best-'mtry' Runs

Description

Given a sweep produced by [missforest_sweep_mtry()] and a table naming the winning 'mtry' for each column, pulls each column out of the run that won it. Different columns may come from different runs.

Usage

```
missforest_impute_by_mtry(sweep, best)
```

Arguments

`sweep`	A named list of [missforest_oob_by_mtry()] results, named by 'mtry' as a character string (as [missforest_sweep_mtry()] builds it).
`best`	A data frame with one row per column, containing at least the columns 'column' and 'mtry' – typically the 'best' element returned by [missforest_sweep_mtry()].

Details

Runs are looked up by **name** ('sweep[[as.character(mtry)]]'), not by position. Positional lookup happens to work only when the swept 'mtry' values are exactly '1:n'; it silently returns the wrong run for any other sweep, such as one starting above 1 or skipping values.

Value

A tibble of imputed columns, one per row of 'best'.

See Also

[missforest_sweep_mtry()]

`missforest_oob_by_mtry` *Run 'missForest' at a Single 'mtry' and Report Variablewise OOB Error*

Description

Fits [missForest::missForest()] once at a given 'mtry', returning both the completed data and a tidy, per-column out-of-bag error table. This is the unit of work swept over by [missforest_sweep_mtry()].

Usage

```
missforest_oob_by_mtry(data, mtry, ntree = 100, maxiter = 10)
```

Arguments

`data`	A data frame or tibble containing the columns to impute. Any identifier or time columns should already be removed; see [missforest_sweep_mtry()]'s 'exclude' argument. Columns that are entirely 'NA' are refused – see Details.
`mtry`	Number of variables randomly sampled as candidates at each split. A single positive integer.
`ntree`	Number of trees per forest. Passed to [missForest::missForest()]. Default '100'.
`maxiter`	Maximum number of imputation iterations. Passed to [missForest::missForest()]. Default '10'.

Details

'missForest' requires every column to be numeric or a factor, so character columns are converted to factors here. The conversion is idempotent – a caller that has already converted its own characters sees no change – which is what lets this single function replace call sites that previously converted at different points.

Columns that are entirely 'NA' are refused with an error. [missForest::missForest()] silently drops them, which both breaks the per-column error table this function returns and would quietly shrink the predictor set that swept 'mtry' values index into. Such columns arise easily – a mapping-driven reader that fills absent source columns with a bare 'NA' produces full-length logical columns that pass name-based schema checks. Drop or fill them before imputing.

Value

A list of two elements:

'ximp' A tibble of the completed (imputed) data.

'oob_error' A tibble with one row per column of 'data', and columns 'column', 'error_type' ('"MSE"' for numeric columns, '"PFC"' for factors), 'error', and 'mtry'.

Reproducibility

'missForest' is stochastic. This function does not set a seed – seeding is the caller's responsibility, and [missforest_sweep_mtry()] handles it in a parallel-safe way. See that function's Reproducibility section.

See Also

[missforest_sweep_mtry()], [missforest_impute_by_mtry()]

Examples

```
if (requireNamespace("missForest", quietly = TRUE)) {
  set.seed(1)
  df <- data.frame(a = c(1, 2, NA, 4, 5), b = c(2, NA, 6, 8, 10))
  res <- missforest_oob_by_mtry(df, mtry = 1)
  res$oob_error
}
```

---

missforest_sweep_mtry *Impute a Data Frame by Sweeping 'missForest' Over 'mtry'*

Description

Runs [missForest::missForest()] once for every candidate 'mtry', then keeps, **for each column independently**, the imputation from whichever 'mtry' minimised that column's out-of-bag error. Columns in the same data frame may therefore be imputed at different 'mtry' values.

Usage

```
missforest_sweep_mtry(
  data,
  exclude = NULL,
  mtry_values = NULL,
  ntree = 100,
  maxiter = 10,
  seed = TRUE,
  max_pct_missing = NULL,
  parallel = TRUE
)
```

Arguments

`data`	A data frame or tibble to impute. After 'exclude' is applied, any remaining column that is entirely 'NA' is refused with an error; see [missforest_oob_by_mtry()]'s Details.
`exclude`	Character vector of columns to hold out of the imputation entirely – typically subject identifiers and time columns. They are removed before fitting and re-attached, unchanged, to the result. Default 'NULL'. A name that is not a column of 'data' is an error, so a misspelt identifier cannot stay in the predictor set unnoticed.
`mtry_values`	Integer vector of 'mtry' values to sweep. Defaults to '1:(p - 1)', where 'p' is the number of columns remaining after 'exclude'.
`ntree, maxiter`	Passed to [missforest_oob_by_mtry()].

`seed` Controls parallel-safe random number generation; passed to [furrr::furrr_options()]. ‘TRUE‘ (the default) draws reproducible streams from the current RNG state, so ‘set.seed()‘ beforehand makes the whole sweep reproducible. A single integer seeds the sweep self-containedly, independent of ambient RNG state. See Reproducibility.

`max_pct_missing`
Optional proportion in ‘(0, 1]‘. Columns missing a greater share than this are held out of the imputation and carried through unimputed, and are reported in ‘excluded_high_missing‘. ‘NULL‘ (the default) imputes every column regardless of how sparse it is. Imputing a column observed in a handful of rows manufactures values rather than recovering them, and nothing downstream can tell the difference; a threshold states that judgement as a rule that carries to the next data set, instead of naming the offending column inline. Note a threshold below ‘1‘ subsumes the all-‘NA‘ refusal, since an all-‘NA‘ column exceeds every threshold.

`parallel` Whether to evaluate the sweep with [furrr::future_map()]. Default ‘TRUE‘. When ‘FALSE‘, runs sequentially via [purrr::map()].

Details

‘best$error‘ is a selection criterion, not an error estimate. ‘best$error‘ is the minimum of OOB errors computed over the swept ‘mtry‘ grid on the same observed cells, with predictors that were themselves iteratively imputed. It is optimistically biased and should be used strictly for model selection, not reported as an unbiased estimate of imputation accuracy.

Value

A list of four elements:

‘imp_data‘ A tibble of the imputed data, with ‘exclude‘ columns re-attached and the original column order restored. Character and logical columns retain their original classes. Integer columns with imputed values are returned as numeric/double because random forest predictions are continuous.

‘oob_error‘ A tibble of every incomplete column’s OOB error at every swept ‘mtry‘.

‘best‘ The winning row per column – the subset of ‘oob_error‘ minimising ‘error‘ within each ‘column‘, ties broken by first occurrence. A column whose error is missing at every ‘mtry‘ (for example one observed in a single row, where ‘missForest‘ reports ‘NaN‘) cannot be ranked: it keeps its first row, with a missing ‘error‘, and a warning names it.

‘excluded_high_missing‘ Columns held out by ‘max_pct_missing‘, carried through unimputed. ‘character(0)‘ when none were.

Reproducibility

‘missForest‘ is stochastic, so an unseeded sweep is not reproducible. In parallel mode the seed **must** reach [furrr::future_map()]’s ‘.options‘; attaching ‘furrr_options()‘ to [future::plan()] instead is silently ignored (‘future‘ warns about an unknown argument and proceeds unseeded). This function always passes ‘seed‘ through to the ‘future_map()‘ call.

When an integer ‘seed‘ is provided, each ‘mtry‘ value ‘m‘ is deterministically seeded via ‘seed + m‘, ensuring that a given ‘mtry‘ produces identical results regardless of the rest of the swept grid. Furthermore, the caller’s ‘.Random.seed‘ state is preserved and restored upon exit for both sequential and parallel modes.

Seeded this way, results are independent of the number of workers and of the ‘future‘ plan, so a Windows ‘multisession‘ run and a Linux ‘multicore‘ run draw identical random streams. Note that identical streams do not by themselves guarantee bit-identical output across *architectures*: tree splits compare floating-point impurity sums, and differences in FMA contraction or BLAS between x86_64 and arm64 can resolve a near-tie differently. Verify cross-architecture agreement rather than assuming it.

The sequential path (‘parallel = FALSE‘) seeds with [set.seed()] when ‘seed‘ is a number, and restores the caller's RNG state on exit. It is internally reproducible but will not match the parallel path, which uses L'Ecuyer-CMRG streams.

Parallel plan

This function never calls [future::plan()] – the appropriate backend differs by machine, and choosing one here would override the caller's. Set a plan before calling, for example ‘future::plan(future::multisession, workers = future::availableCores() - 1)‘.

See Also

[missforest_oob_by_mtry()], [missforest_impute_by_mtry()]

Examples

```
if (requireNamespace("missForest", quietly = TRUE)) {
  df <- data.frame(
    id = 1:8,
    a = c(1, 2, NA, 4, 5, 6, NA, 8),
    b = c(2, NA, 6, 8, 10, 12, 14, NA),
    c = c(5, 4, 3, NA, 1, 2, 3, 4)
  )
  res <- missforest_sweep_mtry(df, exclude = "id", seed = 42, parallel = FALSE)
  res$best
}
```

---

missmap *Missingness Map*

Description

Visualizes missingness across a data frame as a heatmap, either per row/column or aggregated by a grouping column.

Usage

```
missmap(
  df,
  by_column = NULL,
  na_list = NULL,
  row_order = FALSE,
  fill = c("auto", "binary", "count")
)
```

Arguments

df	A data frame or tibble.
by_column	Optional unquoted column name to aggregate missingness by (e.g. a site or subject id) instead of plotting every row individually. Missingness is summed per feature within each group. When row_order = FALSE, both the groups (x-axis) and the features (y-axis) are ordered by descending total missingness, exactly as in the per-row/column view.
na_list	Optional vector of additional values to treat as missing (beyond actual NA).
row_order	Logical; if FALSE (default) rows/features (or, in by_column mode, groups/features) are ordered by how much missingness they have.
fill	In by_column mode, how to color the tiles: "auto" (default) picks a discrete two-level "Missing"/"Present" fill (matching the per-row/column view) when every group has at most one contributing row – in which case the summed missingness count is always 0 or 1 and a count scale would be misleading – and otherwise falls back to a continuous "# missing" gradient. "binary" and "count" force one of those two behaviors regardless of group size. Ignored outside by_column mode.

Value

A ggplot object.

Examples

```
df <- data.frame(
  a = c(1, NA, 3, 4, NA),
  b = c(NA, NA, 3, 4, 5),
  site = c("A", "A", "B", "B", "B")
)
missmap(df)
missmap(df, by_column = site)
```

---

missranger_max_mtry *Largest 'mtry' 'missRanger' Will Accept For a Data Set*

Description

'missRanger' admits a **smaller** 'mtry' than 'missForest' on the same data, and the difference is easy to trip over: exceeding it surfaces as 'ranger''s catch-all '"User interrupt or internal error."', which names neither 'mtry' nor the real bound.

Usage

```
missranger_max_mtry(data)
```

Arguments

data	A data frame or tibble, after any identifier columns have been excluded.

Details

'missForest' predicts every column from every other one, so its bound is simply 'ncol(data) - 1'. 'missRanger' builds its predictor pool up over the first iteration instead. It starts with 'completed', the set of **fully-observed** columns, imputes the targets in increasing order of missingness, and adds each one to 'completed' as it goes. The first target therefore sees only the complete columns, and 'mtry' must fit *that* pool:

* With 'k' usable complete columns, the bound is 'k'. * With none, the first target falls back to univariate imputation and the second sees a single predictor, so the bound is '1' – no matter how wide the data is.

Constant columns are excluded from the count, matching 'missRanger''s own handling of them as features.

The practical consequence: a sweep is only informative when some columns are complete. That is the usual shape of the intended use – imputing a domain's incomplete outputs against its fully-observed inputs – but a frame in which *every* column has missing values admits 'mtry = 1' alone, and there is nothing to sweep.

Value

A single integer: the largest admissible 'mtry', at least '1'.

See Also

[missranger_sweep_mtry()]

Examples

```
# Two complete columns -> mtry may be 1 or 2.
missranger_max_mtry(data.frame(a = c(1, NA, 3), b = c(1, 2, 3), c = c(4, 5, 6)))

# Every column has a gap -> only mtry = 1 is admissible.
missranger_max_mtry(data.frame(a = c(1, NA, 3), b = c(NA, 2, 3), c = c(4, 5, NA)))
```

`missranger_oob_by_mtry` *Run 'missRanger' at a Single 'mtry' and Report Per-Column OOB Error*

Description

Fits [missRanger::missRanger()] once at a given 'mtry', returning both the completed data and a tidy, per-column out-of-bag error table. This is the unit of work swept over by [missranger_sweep_mtry()], and the 'ranger'-based counterpart to [missforest_oob_by_mtry()].

Usage

```
missranger_oob_by_mtry(
  data,
  mtry,
  num.trees = 500,
  pmm.k = 0,
  maxiter = 10,
```

```
  seed = NULL,
  num.threads = 1,
  keep_forests = FALSE
)
```

Arguments

data	A data frame or tibble containing the columns to impute. Any identifier or time columns should already be removed; see [missranger_sweep_mtry()]'s 'exclude' argument. Columns that are entirely 'NA' are refused – see Details.
mtry	Number of variables randomly sampled as candidates at each split. A single positive integer.
num.trees	Number of trees per forest. Passed to [missRanger::missRanger()]. Default '500', that function's own default.
pmm.k	Number of predictive-mean-matching donors. '0' (the default) disables PMM and imputes with the forest prediction directly.
maxiter	Maximum number of imputation iterations. Default '10'.
seed	Integer seed passed to [missRanger::missRanger()], or 'NULL' (default) to leave it unseeded. See Reproducibility.
num.threads	Threads used by 'ranger' *within* one fit. Defaults to '1'; see the Threading section, which explains why the 'ranger' default is the wrong one here.
keep_forests	Whether to retain the fitted forests on the returned object. Default 'FALSE'. See Details.

Details

Error rows cover imputed columns only. 'missForest' run variablewise reports an OOB error for every column; 'missRanger' fits forests only for columns that actually have missing values, so complete columns get no row. [missranger_sweep_mtry()] carries those columns through unchanged.

Errors are not comparable across engines. 'missForest' reports raw MSE and PFC. 'missRanger' reports 'ranger''s scaled OOB prediction error – '1 - R^2' for numeric targets and classification error for categorical ones – where roughly '1' means no better than predicting the mean. Compare 'mtry' values within one engine; do not compare an 'error' column from one against the other.

All-'NA' columns are refused. 'missRanger' does not drop them the way 'missForest' does; it silently returns them still entirely 'NA', producing a frame that looks imputed but is not. Such columns arise easily – a mapping-driven reader that fills absent source columns with a bare 'NA' yields full-length logical columns that pass name-based schema checks.

'keep_forests' is the seam for applying to new data. 'missForest' discards its forests, which is why it cannot be applied to a later batch. 'missRanger' can retain them, making 'predict()' on new data possible. This function only exposes the option; a fit/apply pair built on it is separate work.

Value

A list of two elements, plus a third when 'keep_forests = TRUE':

'ximp' A tibble of the completed (imputed) data.

'oob_error' A tibble with columns 'column', 'error_type', 'error', and 'mtry' – one row per *imputed* column. See Details.

'fit' The 'missRanger' object, when 'keep_forests = TRUE'.

Threading

'ranger' defaults to using every available core. Inside a parallel 'mtry' sweep that nests one thread pool inside another and oversubscribes the machine badly – workers times cores threads competing for the same cores. 'num.threads' therefore defaults to '1' here, leaving parallelism to the sweep. Raise it only when running a single fit on an otherwise idle box.

Reproducibility

Unlike 'missForest', 'missRanger' takes its own 'seed' and uses it to seed the fit directly, so a seeded fit is fully determined by 'seed' regardless of ambient RNG state, worker count, or 'future' plan. That makes seeded 'missranger_*' sweeps reproducible by construction rather than by careful stream management. The architecture caveat still applies: identical seeds do not guarantee bit-identical results across x86_64 and arm64, because split selection compares floating-point impurity sums.

A seeded fit leaves the caller's random number stream as it found it: 'missRanger' re-seeds the global generator, so the generator state is restored when the function exits, and later draws in the caller (a bootstrap loop, say) carry on as if the fit had not run. An unseeded fit draws from the caller's stream as usual.

See Also

[missranger_sweep_mtry()], [missforest_oob_by_mtry()]

Examples

```
if (requireNamespace("missRanger", quietly = TRUE)) {
  df <- data.frame(
    a = c(1, 2, NA, 4, 5, 6, 7, 8),
    b = c(2, NA, 6, 8, 10, 12, 14, 16),
    c = c(5, 4, 3, NA, 1, 2, 3, 4)
  )
  res <- missranger_oob_by_mtry(df, mtry = 1, num.trees = 50, seed = 1)
  res$oob_error
}
```

---

missranger_sweep_mtry *Impute a Data Frame by Sweeping 'missRanger' Over 'mtry'*

Description

Runs [missRanger::missRanger()] once for every candidate 'mtry', then keeps, **for each column independently**, the imputation from whichever 'mtry' minimised that column's out-of-bag prediction error. The 'ranger'-based counterpart to [missforest_sweep_mtry()], with the same return shape so the two engines can be compared on one data set.

Usage

```
missranger_sweep_mtry(
  data,
  exclude = NULL,
```

```
  mtry_values = NULL,
  num.trees = 500,
  pmm.k = 0,
  maxiter = 10,
  seed = NULL,
  num.threads = 1,
  max_pct_missing = NULL,
  parallel = TRUE
)
```

Arguments

data	A data frame or tibble to impute. After ‘exclude‘ is applied, any remaining column that is entirely ‘NA‘ is refused with an error.
exclude	Character vector of columns to hold out of the imputation entirely – typically subject identifiers and time columns. They are removed before fitting and re-attached, unchanged, to the result. Default ‘NULL‘. A name that is not a column of ‘data‘ is an error, so a misspelt identifier cannot stay in the predictor set unnoticed.
mtry_values	Integer vector of ‘mtry‘ values to sweep. Defaults to ‘1:missranger_max_mtry(data)‘. Note this is **not** ‘1:(p - 1)‘, the ‘missForest‘ bound – see [missranger_max_mtry()] for why, and expect a shorter sweep than [missforest_sweep_mtry()] performs on the same data.
num.trees, pmm.k, maxiter, num.threads	Passed to [missranger_oob_by_mtry()].
seed	Integer seed passed to every fit in the sweep, or ‘NULL‘ (default) for an unseeded sweep. See Reproducibility.
max_pct_missing	Optional proportion in ‘(0, 1]‘. Columns missing a greater share than this are held out of the imputation and carried through unimputed, and are reported in ‘excluded_high_missing‘. ‘NULL‘ (the default) imputes every column regardless of how sparse it is. Imputing a column observed in a handful of rows manufactures values rather than recovering them, and nothing downstream can tell the difference; a threshold states that judgement as a rule that carries to the next data set, instead of naming the offending column inline. Note a threshold below ‘1‘ subsumes the all-‘NA‘ refusal, since an all-‘NA‘ column exceeds every threshold.
parallel	Whether to evaluate the sweep with [furrr::future_map()]. Default ‘TRUE‘. When ‘FALSE‘, runs sequentially via [purrr::map()].

Details

‘best$error‘ is a selection criterion, not an error estimate. ‘best$error‘ is the minimum of OOB prediction errors computed over the swept ‘mtry‘ grid on the same observed cells, with predictors that were themselves iteratively imputed. It is optimistically biased and should be used strictly for model selection, not reported as an unbiased estimate of imputation accuracy.

Value

A list of four elements, matching [missforest_sweep_mtry()]:

‘imp_data‘ A tibble of the imputed data, with ‘exclude‘ columns re-attached and the original column order restored. Character and logical columns retain their original classes. Integer columns with imputed values are returned as numeric/double because random forest predictions are continuous.

‘oob_error‘ A tibble of every imputed column’s OOB error at every swept ‘mtry‘.

‘best‘ The winning row per imputed column. A column whose error is missing at every ‘mtry‘ cannot be ranked: it keeps its first row, with a missing ‘error‘, and a warning names it. When no column has missing values nothing is scored, and ‘oob_error‘ and ‘best‘ are empty tables.

‘excluded_high_missing‘ Columns held out by ‘max_pct_missing‘, carried through unimputed. ‘character(0)‘ when none were.

Reproducibility

The same ‘seed‘ is passed to every fit in the sweep. That is deliberate: holding the seed fixed across ‘mtry‘ values is a common-random-numbers comparison, so differences in OOB error reflect ‘mtry‘ rather than sampling noise between runs. Because ‘missRanger‘ seeds itself, a seeded sweep is reproducible independently of the ‘future‘ plan and the number of workers – the property a mixed Windows/Linux/arm64 fleet needs – without depending on L’Ecuyer stream management. ‘furrr‘’s own parallel-safe seeding is still enabled, which covers the unseeded case. A seeded sequential sweep (‘parallel = FALSE‘) restores the caller’s random number state when each fit finishes, so it does not disturb later draws in the calling code. When ‘seed‘ is an integer, caller’s ‘.Random.seed‘ is restored on exit for both sequential and parallel modes.

Parallel plan

This function never calls [future::plan()] – the appropriate backend differs by machine, and choosing one here would override the caller’s. Set a plan before calling. Note that ‘num.threads‘ defaults to ‘1‘ so that ‘ranger‘’s own threading does not nest inside the sweep; see [missranger_oob_by_mtry()]’s Threading section.

See Also

[missforest_sweep_mtry()] for the ‘missForest‘ engine.

Examples

```
if (requireNamespace("missRanger", quietly = TRUE)) {
  df <- data.frame(
    id = 1:12,
    a = c(1, 2, NA, 4, 5, 6, NA, 8, 9, 10, 11, NA),
    b = c(2, 4, 6, NA, 10, 12, 14, NA, 18, 20, NA, 24),
    c = c(5, 4, 3, NA, 1, 2, 3, 4, 5, NA, 3, 2)
  )
  res <- missranger_sweep_mtry(
    df, exclude = "id", num.trees = 50, seed = 42, parallel = FALSE
  )
  res$best
}
```

`multiple_t_test` *Multiple T-Tests Against One Classifier*

Description

Runs `single_t_test` for every (or a chosen set of) numeric column against a single binary classifier.

Usage

```
multiple_t_test(
  .data,
  .var_list = setdiff(names(dplyr::select(.data, dplyr::where(is.numeric))), c(.class,
    .id)),
  .class,
  .id = NULL,
  alternative = "two.sided",
  conf.level = 0.95,
  ...
)
```

Arguments

`.data`	A data frame or tibble.
`.var_list`	Character vector of column names to test (default: every numeric column in `.data` except `.class` and `.id`).
`.class`	Name (string) of a binary classification column.
`.id`	Name (string, optional) of a subject/record identifier column, passed to `single_t_test` (see there, for `paired = TRUE`). Like `.class`, it is left out of the default `.var_list`.
`alternative`	One of `"two.sided"` (default), `"greater"`, or `"less"`.
`conf.level`	Confidence level for the interval.
`...`	Additional arguments passed to `single_t_test` (and on to `t.test`) — e.g. `paired`.

Value

A tibble with one row per tested variable.

Examples

```
mtcars |>
  dplyr::mutate(am = factor(am)) |>
  multiple_t_test(.class = "am")
```

my_summary_table *Summary Table Function*

Description

Summary Table Function

Usage

```
my_summary_table(
  data,
  var,
  na.rm = FALSE,
  data.output = TRUE,
  table.output = FALSE
)
```

Arguments

data	a dataframe or tibble
var	variable to summarize
na.rm	should NA values be removed, TRUE or FALSE, default is FALSE
data.output	should the data be output, TRUE or FALSE, default is TRUE
table.output	should the table be output, TRUE or FALSE, default is FALSE

Value

either a dataframe or readable table

Examples

```
mtcars |>
  dplyr::group_by(cyl) |>
  my_summary_table(mpg)

mtcars |>
  dplyr::group_by(cyl) |>
  my_summary_table(mpg, table.output = TRUE)
```

nested_cv_coxnet *Nested Cross-Validation of a Penalized Cox Model*

Description

Estimates how well the whole tuning procedure of [cv_coxnet()] generalizes. For each outer split of an [rsample::nested_cv()] object: ‘mixture‘ and ‘penalty‘ are chosen by [cv_coxnet()] on that split's inner resamples; the model is refit on the outer analysis set with those settings; and the refit is scored on the outer assessment set, which played no part in tuning.

Usage

```
nested_cv_coxnet(
  object,
  preprocessor,
  subject_id = NULL,
  group = NULL,
  rule = c("min", "1se"),
  mixture = 1,
  penalty = NULL,
  metrics = NULL,
  eval_time = NULL,
  metric = NULL,
  covariates = c("path", "baseline"),
  trunc = 0.05,
  parallel = FALSE,
  importance = c("none", "loco_mp"),
  check_subject_overlap = TRUE,
  tune_method = c("grid", "race_anova", "race_win_loss"),
  ...
)

## S3 method for class 'nested_cv_coxnet'
collect_metrics(x, ..., summarize = TRUE)
```

Arguments

object	An [rsample::nested_cv()] object.
preprocessor	A formula with a ‘Surv()‘ outcome, or a [recipes::recipe()] whose outcome is a ‘Surv‘ column.
subject_id, group	Column names, as in [cv_coxnet()].
rule	Use ‘"min"‘ (‘lambda.min‘) or ‘"1se"‘ (‘lambda.1se‘) from each inner cross-validation.
mixture	Numeric vector of elastic-net mixing values to try.
penalty	Optional decreasing penalty path; by default glmnet’s path for each ‘mixture‘.
metrics	A [yardstick::metric_set()] of survival metrics. Defaults to the integrated Brier score, concordance, time-specific Brier score, and time-dependent ROC AUC.
eval_time	Evaluation times, shared by all splits. Defaults to deciles of the event times in the full data.
metric	Name of the metric used to choose settings; the first metric in ‘metrics‘ by default.
covariates	‘"path"‘ or ‘"baseline"‘; see Details.
trunc	Lower bound for the censoring probability in the Graf weights.
parallel	If ‘TRUE‘, fit folds with [furrr::future_map()]; set a [future::plan()] first.
importance	Method for evaluating feature importance on the outer analysis sets. Options are ‘"none"‘ (default) or ‘"loco_mp"‘ (Leave-One-Covariate-Out with MiniPatch ensembles). [cbe_loco_mp_coxnet()] is run on each outer analysis set at the ‘mixture‘ and penalty chosen there, with the same glmnet arguments (see ‘...‘). It draws its minipatches by subject, so ‘group‘ does not apply to it, and a warning says so when ‘group‘ is set.

`check_subject_overlap`
: If ‘TRUE‘ (the default), stop when an outer split, or an inner resample, puts the same subject (or, when ‘group‘ is set, the same group) in both its analysis and its assessment set; see Details and [cv_coxnet()]. ‘FALSE‘ skips the check.

`tune_method`
: Method for tuning hyperparameters across the inner resamples: ‘"grid"‘ (default, standard [cv_coxnet()] grid search), ‘"race_anova"‘ ([finetune::tune_race_anova()] racing), or ‘"race_win_loss"‘ ([finetune::tune_race_win_loss()] racing; needs BradleyTerry2).

`...`
: Further arguments passed to [glmnet::glmnet()], such as ‘cox.ties‘, ‘standardize‘, ‘penalty.factor‘, or ‘nlambda‘. With ‘importance = "loco_mp"‘ they are passed to [cbe_loco_mp_coxnet()] as well, so the minipatch models are fit like the tuned ones; arguments that [cbe_loco_mp_coxnet()] sets itself, such as ‘B‘, are not forwarded.

`x`
: A ‘nested_cv_coxnet‘ object.

`summarize`
: For ‘collect_metrics()‘: ‘TRUE‘ for the mean and standard error over outer splits, ‘FALSE‘ for each split’s metrics.

Details

Build ‘object‘ with grouped resampling at both levels, e.g. ‘nested_cv(data, outside = group_vfold_cv(group = subject_id), inside = group_vfold_cv(group = subject_id))‘, so a subject’s start/stop rows stay together. This is checked: if a subject (or, when ‘group‘ is set, a group) is in both the analysis and the assessment set of an outer split or of any inner resample, ‘nested_cv_coxnet()‘ stops before fitting anything, because the model would be scored on subjects it was fit on. Out-of-bag assessment sets of bootstraps pass. ‘check_subject_overlap = FALSE‘ skips the check, for overlap that is deliberate.

The outer assessment set is scored as in [cv_coxnet()]: a subject with a missing predictor after preprocessing, such as a factor level the outer analysis set never saw, is left out with a warning, and an outer split with no subject left is an error.

The outer metrics describe the procedure, not one final model: the chosen settings can differ between outer splits. To fit a final model, run [cv_coxnet()] on all the data.

Value

A tibble, of class ‘nested_cv_coxnet‘, with one row per outer split: ‘id‘, the chosen ‘mixture‘ and ‘penalty‘, ‘.metrics‘ (outer assessment-set metrics), ‘.coefs‘ (coefficients of the refit), and ‘.inner‘ (the inner cross-validation’s summarized metrics). If ‘importance = "loco_mp"‘, includes ‘.importance‘ with tidy LOCO-MP statistical inference. ‘tune::collect_metrics()‘ averages ‘.metrics‘ over outer splits.

See Also

[cv_coxnet()], [coxnet()], [cbe_loco_mp_coxnet()], [tune_race_survival()]

Examples

```
if (requireNamespace("glmnet", quietly = TRUE) &&
    requireNamespace("survival", quietly = TRUE) &&
    requireNamespace("rsample", quietly = TRUE) &&
    requireNamespace("yardstick", quietly = TRUE)) {
  set.seed(1)
  long <- do.call(rbind, lapply(1:90, function(i) {
    k <- sample(1:3, 1)
```

```
    stops <- cumsum(stats::runif(k, 2, 6))
    risk <- stats::rnorm(1)
    data.frame(subject_id = i, tstart = c(0, utils::head(stops, -1)), tstop = stops,
               status = c(rep(0, k - 1), stats::rbinom(1, 1, stats::plogis(risk))),
          x1 = risk + stats::rnorm(k, sd = 0.2), x2 = stats::rnorm(k), x3 = stats::rnorm(k))
  }))
  folds <- rsample::nested_cv(
    long,
    outside = rsample::group_vfold_cv(group = subject_id, v = 3),
    inside = rsample::group_vfold_cv(group = subject_id, v = 3)
  )
  res <- nested_cv_coxnet(
    folds, survival::Surv(tstart, tstop, status) ~ x1 + x2 + x3,
    subject_id = "subject_id", mixture = c(0.5, 1),
    metrics = yardstick::metric_set(yardstick::brier_survival_integrated,
                                    yardstick::concordance_survival),
    nlambda = 20, cox.ties = "breslow"
  )
  res
}
```

`nested_cv_joint_model` *Nested Cross-Validation for Joint Models*

Description

Scores the `joint_model` pipeline on the outer splits of an `nested_cv` object. As in `cv_joint_model`, each outer split's model is fit on its analysis set and scored on its assessment set one row per subject, and the `coxnet` model of a start/stop outcome is scored along each subject's covariate path (`covariates`), as `cv_coxnet` does. The inner resamples are not used to fit anything: the penalty of each outer split's `coxnet` model is tuned by the internal cross-validation of `joint_model()` (unless `penalty` is given), and the status and time models choose theirs by `cv.glmnet`.

Usage

```
nested_cv_joint_model(
  object,
  outcome = survival::Surv(time, status) ~ .,
  subject_id = NULL,
  engine = c("glmnet", "baguette", "stacks"),
  calibration = TRUE,
  parallel = FALSE,
  covariates = c("path", "baseline"),
  check_subject_overlap = TRUE,
  tune_method = c("none", "race_anova", "race_win_loss"),
  ...
)
```

Arguments

`object`	An `rsample::nested_cv` object.
`outcome`	Survival formula with a `Surv()` outcome.

subject_id	Subject identifier column. Required for start/stop outcomes.
engine	Modeling engine: "glmnet", "baguette", or "stacks" (the "baguette" models plus a stored Cox meta-learner whose output predict() adds as extra columns; see joint_model).
calibration	Logical; whether to calibrate status predictions (default TRUE).
parallel	Logical; whether to run outer splits in parallel.
covariates	"path" or "baseline"; see cv_joint_model.
check_subject_overlap	If TRUE (the default), stop when an outer split, or an inner resample, puts the same subject in both its analysis and its assessment set; see Details. FALSE skips the check.
tune_method	Method for tuning the Cox component penalty on the inner resamples: "none" (default, uses internal CV of joint_model()), "race_anova" (tune_race_anova racing), or "race_win_loss" (tune_race_win_loss racing, which needs **BradleyTerry2**). With racing, the penalty found on each outer split's inner resamples replaces any penalty given in ....
...	Additional arguments passed to joint_model.

Details

A counting-process (start/stop) outcome needs subject_id, and the outer splits and every inner resample are checked before anything is fit: if a subject is in both the analysis and the assessment set of one of them, the function stops, as cv_joint_model does for supplied resamples. Build object with grouped resampling at both levels, e.g. nested_cv(data, outside = group_vfold_cv(group = subject_id), inside = group_vfold_cv(group = subject_id)). Out-of-bag assessment sets of bootstraps pass. check_subject_overlap = FALSE skips the check.

Value

An S3 object of class c("nested_cv_joint_model", "tbl_df"): outer_id, model, and ibs, one row per outer split and model.

---

normalize_safely *Normalize File Paths Without Failing*

Description

A vectorized [normalizePath()] that never errors or warns: blank and 'NA' inputs become 'NA', and a path that cannot be normalized comes back as it was. Paths need not exist. Separators are forward slashes on every platform.

Usage

```
normalize_safely(x)
```

Arguments

x	Character vector of paths.

Value

A character vector of normalized paths, named by the inputs.

See Also

[scan_data_io()]

Examples

```
normalize_safely(c("data/../data/file.csv", "", NA))
```

`one_vs_rest_t_test` *One-vs-Rest T-Tests Across a Multi-Level Factor*

Description

For a classifier with more than two levels, dichotomizes each level against all others in turn and runs `single_t_test`.

Usage

```
one_vs_rest_t_test(
  .data,
  .var,
  .class,
  .id = NULL,
  paired = FALSE,
  alternative = "two.sided",
  conf.level = 0.95
)
```

Arguments

`.data`	A data frame or tibble.
`.var`	Name (string) of the continuous column to test.
`.class`	Name (string) of a classification column (2+ levels).
`.id`	Name (string, optional) of a subject/record identifier column; see `single_t_test`. Only meaningful when `paired = TRUE`, and then only for a two-level `.class`: with three or more levels the pooled "rest" group holds several observations per id, which is an error.
`paired`	Logical; paired t-test.
`alternative`	One of `"two.sided"` (default), `"greater"`, or `"less"`.
`conf.level`	Confidence level for the interval.

Value

A tibble with one row per level of `.class`.

Examples

```
one_vs_rest_t_test(iris, "Sepal.Length", "Species")
```

package_deliverables *Package Pipeline Deliverables into a Structured Zip Archive*

Description

Traverses computational pipeline objects (or explicit file lists), gathers all rendered reports and data deliverables, verifies existence on disk, organizes them into stage-prefixed folders, and builds a distribution ZIP file.

Usage

```
package_deliverables(
  pipeline_objects,
  output_formats = c("pdf", "docx"),
  data_deliverables = list(),
  zip_path = NULL,
  include_pipeline_graph = TRUE
)
```

Arguments

pipeline_objects	A list of `FilePath` / `FileOutputs` objects (e.g. from the compute graph). The `stage` of each `FileOutputs` becomes a folder name in the archive: path separators and the characters a file name cannot hold on Windows become `-`, and leading or trailing dots, dashes and spaces are dropped (`"../x"` gives `"x"`).
output_formats	Character vector of report formats to include (`"pdf"`, `"docx"`, `"html"`, or `"all"`).
data_deliverables	Optional list or vector of `FilePath` objects or file paths for data spreadsheets/RDS files.
zip_path	Destination path for the generated zip archive. If NULL, defaults to a timestamped archive. Deliverables that share a file name within a folder are kept, the later ones with a numeric suffix (and a warning), rather than overwriting each other.
include_pipeline_graph	Logical; whether to include static/interactive pipeline DAG graphs (default: TRUE).

Value

Absolute path to the generated zip file.

parse_here_call_vec *Resolve 'here::here()' Calls Found in Code Text*

Description

For each string, finds the first 'here::here(...)' call and evaluates it with its quoted string arguments, giving the path the call builds in the current project. Calls whose arguments are not all string literals resolve from the literals alone.

Usage

```
parse_here_call_vec(x)
```

Arguments

x	Character vector of code lines.

Value

A character vector the length of 'x': the resolved path, or 'NA' where a line has no 'here::here()' call with quoted arguments.

See Also

[scan_data_io()], [find_code()]

Examples

```
parse_here_call_vec(c("x <- readRDS(here::here('data', 'x.rds'))", "no call"))
```

pca_biplot *PCA Loadings Biplot*

Description

A classic PCA biplot: the observation scores (pca_model$x) overlaid with variable loading vectors (pca_model$rotation) drawn as labeled arrows from the origin. Observations can optionally be colored by a categorical grouping variable with 95% concentration ellipses.

Usage

```
pca_biplot(
  pca_model,
  x = 1,
  y = 2,
  group = NULL,
  ellipse = FALSE,
  title = "PCA Biplot",
  percent = FALSE
)
```

Arguments

`pca_model`	A `prcomp` object.
`x, y`	Which principal components to plot on the x/y axes (default 1, 2).
`group`	Optional categorical vector (e.g. `iris$Species`) to color observations by.
`ellipse`	Logical (default `FALSE`); if `TRUE` and `group` is provided, draws 95% confidence/concentration ellipses around each group.
`title`	Optional plot title (default `"PCA Biplot"`).
`percent`	Logical (default `FALSE`); if `TRUE`, appends percent variance explained to axis labels.

Details

Loading vectors are unit-scale by construction and would be invisible next to the score cloud if plotted as-is, so they are rescaled so that their maximum extent is 80% of the score cloud's maximum extent (a standard biplot convention) before being drawn.

Value

A ggplot object.

Examples

```
pca_model <- prcomp(iris[, 1:4], center = TRUE, scale. = TRUE)
pca_biplot(pca_model, group = iris$Species, ellipse = TRUE)
```

`pca_eqns` *PCA Equations*

Description

Writes out each principal component as a linear equation in the original features (abbreviated `f1`, `f2`, ... — see the returned `labels` table for what each abbreviation means).

Usage

```
pca_eqns(PC_mod, precision = 3)
```

Arguments

`PC_mod`	A `prcomp` object.
`precision`	Digits to round loadings to (default 3).

Value

A list with two tibbles: `eqns` (one row per component, with its equation as text) and `labels` (feature-number-to-name key).

Examples

```
pca_model <- prcomp(mtcars, center = TRUE, scale. = TRUE)
pca_eqns(pca_model)$eqns
pca_eqns(pca_model)$labels
```

pca_feature_loading_heatmap *PCA Feature-Loading Heatmap*

Description

Heatmap of each original feature's loading onto each principal component.

Usage

```
pca_feature_loading_heatmap(pca_model)
```

Arguments

pca_model	A prcomp object.

Value

A ggplot object.

Examples

```
pca_model <- prcomp(mtcars, center = TRUE, scale. = TRUE)
pca_feature_loading_heatmap(pca_model)
```

pca_loading_diff *Difference in PCA Loadings Between Two Fits*

Description

Compares the variable loadings of two independently-fit prcomp objects on the same set of variables (e.g. the same domain's data fit at baseline vs. at a later timepoint), matching components positionally (both fits' PC1, both fits' PC2, ...).

Usage

```
pca_loading_diff(pca_baseline, pca_comparison, n_components = NULL)
```

Arguments

pca_baseline	A prcomp object treated as the reference.
pca_comparison	A prcomp object to compare against pca_baseline, fit on the same (or overlapping) variables.
n_components	Number of leading components to compare. Defaults to NULL, meaning all components shared by both fits.

Details

PCA loading vectors are only unique up to sign: a component can flip orientation between two otherwise-equivalent fits without changing the pattern it represents. Naively differencing loadings would then show a spuriously large change (up to roughly double the loading) for a component that hasn't meaningfully changed at all. To avoid this, for each shared component `pca_comparison`'s loading vector is sign- aligned to `pca_baseline`'s: it is flipped (multiplied by -1) if doing so reduces the total absolute difference across variables relative to leaving it as-is. The difference is then computed as (sign-aligned comparison) minus baseline.

Variables are matched by name (the rownames of `$rotation`). If the two fits were built on different variable sets, only the intersection is used; no error is raised.

Value

A tibble with one row per shared variable (`feature` column) and one column per compared component (`PC1`, `PC2`, ...) holding the sign-aligned difference (comparison minus baseline).

Examples

```
set.seed(1)
baseline <- prcomp(mtcars, center = TRUE, scale. = TRUE)
comparison <- prcomp(mtcars[sample(nrow(mtcars)), ], center = TRUE, scale. = TRUE)
pca_loading_diff(baseline, comparison)
```

`pca_loading_diff_heatmap`

Heatmap of PCA Loading Differences Between Two Fits

Description

Renders the output of `pca_loading_diff` as a heatmap (feature by component), using the same visual language as `pca_feature_loading_heatmap`: a diverging fill scale centered at zero, so components/variables with little sign-aligned change are white and larger changes in either direction stand out in blue or red.

Usage

```
pca_loading_diff_heatmap(pca_baseline, pca_comparison, n_components = NULL)
```

Arguments

`pca_baseline`	A `prcomp` object treated as the reference.
`pca_comparison`	A `prcomp` object to compare against `pca_baseline`, fit on the same (or overlapping) variables.
`n_components`	Number of leading components to compare. Defaults to `NULL`, meaning all components shared by both fits.

Value

A ggplot object.

Examples

```
set.seed(1)
baseline <- prcomp(mtcars, center = TRUE, scale. = TRUE)
comparison <- prcomp(mtcars[sample(nrow(mtcars)), ], center = TRUE, scale. = TRUE)
pca_loading_diff_heatmap(baseline, comparison)
```

---

pca_percent_var_explained *Percent Variance Explained by Each Principal Component*

Description

The percentages come from the component standard deviations (`pca_model$sdev`), as in `pca_scree_plot`, and are shares of the total variance. A fit truncated with `rank.` or `tol` still stores every standard deviation, so all components are shown.

Usage

```
pca_percent_var_explained(pca_model)
```

Arguments

`pca_model`	A `prcomp` object.

Value

A ggplot object showing per-component and cumulative variance explained.

Examples

```
pca_model <- prcomp(mtcars, center = TRUE, scale. = TRUE)
pca_percent_var_explained(pca_model)
```

---

pca_plot *Plot a PCA Fit*

Description

One entry point to this package's PCA plots.

Usage

```
pca_plot(
  pca_model,
  type = c("variance", "heatmap", "bi", "biplot", "scree", "circle"),
  ...
)
```

Arguments

pca_model	A prcomp object. (Not ‘x‘, which would capture the ‘x‘ component argument of ‘type = "bi"‘ and ‘"biplot"‘.)
type	One of "variance" (pca_percent_var_explained), "heatmap" (pca_feature_loading_heatmap "bi" (plot_pca_bi), "biplot" (pca_biplot), "scree" (pca_scree_plot), or "circle" (pca_variables_plot).
...	Passed on to the underlying plot function (needed for type = "bi", which requires newdata and column; and optionally used by type = "bi" or "biplot" to pick components with x/y).

Details

This is an ordinary function rather than a ‘plot()‘ method: **stats** already registers ‘plot()‘ for ‘prcomp‘ objects (a scree plot), and a package that replaces another package’s method for a class it doesn’t own changes ‘plot()‘ for everyone who loads it. Call ‘pca_plot(x, type =)‘, or [stats::screeplot()] for the base scree plot.

Value

A ggplot object.

Examples

```
pca_model <- prcomp(mtcars, center = TRUE, scale. = TRUE)
pca_plot(pca_model, type = "variance")
pca_plot(pca_model, type = "biplot", x = 1, y = 3)
```

---

pca_scree_plot *PCA Scree Plot*

Description

Scree plot displaying the eigenvalues or proportion of variance explained by each principal component, with an optional horizontal reference line at eigenvalue = 1 (Kaiser-Guttman criterion).

Usage

```
pca_scree_plot(
  pca_model,
  metric = c("eigenvalue", "variance"),
  kaiser = TRUE,
  title = "PCA Scree Plot"
)
```

Arguments

pca_model	A prcomp object or output from proc_pca.
metric	Which metric to plot: "eigenvalue" (default, with Kaiser line at 1) or "variance" (percent of variance explained).
kaiser	Logical (default TRUE); draw horizontal dashed line at eigenvalue = 1.
title	Optional plot title.

Value

A ggplot object.

Examples

```
pca_model <- prcomp(iris[, 1:4], center = TRUE, scale. = TRUE)
pca_scree_plot(pca_model)
```

`pca_variables_plot` *PCA Variable Correlation Circle*

Description

Plots the projection of variables onto the unit circle for two principal components, showing correlations between original variables and principal components.

Usage

```
pca_variables_plot(
  pca_model,
  x = 1,
  y = 2,
  title = "Variables Correlation Circle",
  data = NULL
)
```

Arguments

`pca_model`	A `prcomp` object or output from `proc_pca`.
`x, y`	Which principal components to plot (default 1, 2).
`title`	Optional plot title (default `"Variables Correlation Circle"`).
`data`	Optional data frame or matrix holding the original variables (numeric, with a column for each variable of `pca_model`, matched by name). When given, the arrows show `cor(data, scores)`, the correlations between the variables and the component scores of `data`, whatever the scaling of `pca_model`, and the unscaled warning is not given. Default `NULL`: loading times standard deviation.

Details

Each arrow ends at the variable's loading times the component's standard deviation. For a PCA of standardised variables (`prcomp(scale. = TRUE)`, a correlation-matrix PCA) that is the correlation between the variable and the component, so every arrow lies inside the unit circle. For a PCA of unscaled variables (the `prcomp` default, `scale. = FALSE`) it is a covariance in the units of the data, which can exceed 1, and the function warns. A `prcomp` object does not keep the data it was fit to, so either refit with `prcomp(scale. = TRUE)` or give the original variables as `data`, which computes the correlations directly.

Fits truncated with `rank.` or `tol` are supported: the percent of variance on the axis labels is still the share of the total variance.

Value

A ggplot object.

Examples

```
pca_model <- prcomp(iris[, 1:4], center = TRUE, scale. = TRUE)
pca_variables_plot(pca_model)

# An unscaled PCA needs the data to draw correlations.
pca_unscaled <- prcomp(iris[, 1:4])
pca_variables_plot(pca_unscaled, data = iris[, 1:4])
```

---

pdf_to_rtf *Convert a PDF's Text to Rich Text Format (RTF)*

Description

Extracts the text layer of a PDF with [pdftools::pdf_text()] and writes it to an RTF file in a monospaced font, one RTF page per PDF page, so the column alignment of text tables (such as SAS listings) is kept.

Usage

```
pdf_to_rtf(
  pdf,
  rtf = sub("\\.pdf$", ".rtf", pdf, ignore.case = TRUE),
  font_size = 10,
  overwrite = TRUE
)
```

Arguments

pdf	Path to the input '.pdf' file.
rtf	Path of the '.rtf' file to write. Defaults to 'pdf' with its extension replaced by '.rtf'. It must not be the PDF itself, however the path is spelled ('./report.pdf', a different case on Windows), and it must not end in '.pdf': either would overwrite a PDF with RTF text.
font_size	Font size in points.
overwrite	If 'FALSE', error when 'rtf' already exists.

Details

Only text is converted: images, fonts, and vector graphics are not. A scanned PDF with no text layer gives empty pages. Non-ASCII characters are written as RTF Unicode escapes, so the file is plain ASCII and opens the same in any locale.

Value

The path to 'rtf', invisibly.

See Also

[create_toc_from_sas_pdf()]

Examples

```
pdf <- system.file("templates", "example.pdf", package = "TempleCBE")
rtf <- pdf_to_rtf(pdf, tempfile(fileext = ".rtf"))
readLines(rtf, n = 3)
```

`pformat` *Format p-values for Biostatistical and Clinical Reporting*

Description

Formats numeric p-values into clean, publication-ready strings (e.g. `"p = 0.024"`, `"p < 0.001"`, or `"0.024"`).

Usage

```
pformat(p, accuracy = 0.001, add_p = TRUE, digits = NULL)

cbe_pformat(p, accuracy = 0.001, add_p = TRUE, digits = NULL)
```

Arguments

`p`	Numeric p-value or vector of p-values.
`accuracy`	Numeric precision threshold passed to `label_pvalue` (default 0.001).
`add_p`	Logical; if `TRUE` (the default), prepends `"p = "`, `"p < "`, or `"p > "`. If `FALSE`, returns only the formatted numeric string.
`digits`	Optional integer number of decimal places (overrides `accuracy` as `10^(-digits)`).

Value

A character vector of formatted p-values.

Examples

```
pformat(0.0241)
pformat(0.0001)
pformat(0.852, add_p = FALSE)
```

pipeline_config *Get or set global pipeline configuration options*

Description

Decouples project-specific parameters (study name, stage display labels, stage color palettes) from core pipeline mechanics and classes.

Usage

```
pipeline_config(
  study_name = NULL,
  stage_labels = NULL,
  stage_colors = NULL,
  default_formats = NULL
)
```

Arguments

study_name	Character string for the main study title.
stage_labels	Named character vector mapping raw stage identifiers to human-readable labels.
stage_colors	Named list or character vector mapping stage identifiers to hex colors.
default_formats	Optional character vector of default output formats for pipeline renderers (e.g. c("pdf", "docx")). When set, this overrides the package default across render and create_qmd_renderer.

Value

A list containing the current configuration options.

pipeline_summary *Produce an executive summary of the computational pipeline*

Description

Summarizes total nodes, edges, staleness status, artifact role counts, and a stage-by-role breakdown table.

Usage

```
pipeline_summary(object)

summarize_pipeline(object)
```

Arguments

object	A tbl_graph, igraph, or list of FilePath objects.

Value

Invisibly returns a tibble of the stage breakdown.

`pi_anonymizer` *Investigator Name Pseudonymization Helpers*

Description

Helpers that generate synthetic surnames and pseudonym tokens, and replace investigator (PI) names with them in reports and shared output. They accept a single name or a character vector, such as a data frame column.

Scope and limits

These functions pseudonymize **investigator names only**. They do not find, remove, or alter dates, record or subject numbers, locations, free text, or any other identifier, and they are not a general de-identification tool. Replacing investigator names does not by itself make a dataset safe to share.

Pseudonyms are not guaranteed to be unique: two different names can map to the same surname (`method = "hmac_surname"`), and short tokens (a small `n_chars`) collide more often.

Secret key

The deterministic methods (`"hmac_token"` and `"hmac_surname"`) are only as private as their key. Anyone who knows the key, or who can guess it, can recompute every pseudonym from a list of candidate names. The functions therefore stop unless a key is supplied, either through the `key` argument or, preferably, the `TEMPLECBE_SECRET_KEY` environment variable. Keep the key out of scripts and repositories: put it in your user '`.Renviron`' (open it with `usethis::edit_r_environ()`) and restart R:

```
TEMPLECBE_SECRET_KEY=paste-a-long-random-string-here
```

A random key can be made with `paste(as.character(openssl::rand_bytes(32)), collapse = "")`. Use the same key every time to get the same pseudonyms; a lost key cannot be recovered, and a changed key changes every pseudonym.

For a quick, non-private run you can pass `allow_default_key = TRUE`, which falls back to a public built-in key and warns that the output is NOT secret. Methods `"token"` (random tokens kept in a private mapping file) and `"synthetic"` need no key.

See Also

`anonymize_pi`, `generate_pseudonym_token`, `generate_pi_names`, `generate_last_names`

plot.features_percent_miss

Plot method for features_percent_miss objects

Description

Creates a bar chart visualizing feature missingness percentages for objects produced by `features_percent_miss`.

Generates a horizontal bar chart visualizing the percentage of missing values per column.

Usage

```
## S3 method for class 'features_percent_miss'
plot(x, top_n = NULL, ...)

plot_features_percent_miss(data, top_n = NULL)
```

Arguments

`x`	An object of class `features_percent_miss`.
`top_n`	Optional integer to limit to the top N features with highest missingness.
`...`	Additional arguments (currently unused).
`data`	A data frame, tibble, or the output of 'features_percent_miss()'.

Value

A ggplot object representing feature missingness.

A ggplot object.

Examples

```
res <- features_percent_miss(mtcars)
plot(res)

df <- data.frame(a = c(1, 2, NA, 4), b = c(NA, NA, 3, 4))
plot_features_percent_miss(df)
```

plot_categorical_associations

Plot Categorical Association Matrix Using corrplot

Description

Computes pairwise associations (Cramer's V or -log10 p-values) across all categorical/factor columns in a data frame and renders an association matrix plot via **corrplot**. This complements `correlation_plot` for numeric variables.

Usage

```
plot_categorical_associations(
  data,
  cols = NULL,
  method = c("cramer_v", "p_value"),
  title = "Categorical Association Matrix",
  fill_colors = c("#FFFFFF", "#9D2235"),
  tl.cex = 0.8,
  number.cex = 0.75,
  show_coef = TRUE,
  ...
)
```

Arguments

`data`	A data frame or tibble.
`cols`	Optional character vector of column names to include. If `NULL`, all factor and character columns are selected.
`method`	Association metric: `"cramer_v"` (Cramer's V correlation, default) or `"p_value"` ($-\log_{10}(p)$). For `"p_value"`, 2x2 pairs use the exact test of `cbe_exact2x2` and larger tables use the asymptotic Pearson chi-squared test, so every pair is on the same deterministic scale (a simulated Fisher p-value cannot fall below `1 / (B + 1)`). Values are capped at 16.
`title`	Plot title (defaults to `"Categorical Association Matrix"`).
`fill_colors`	Character vector for the color gradient (defaults to Temple palette).
`tl.cex`	Label character expansion (default 0.8).
`number.cex`	Numeric value character expansion inside circles (default 0.75).
`show_coef`	Logical (default `TRUE`); whether to print coefficients.
`...`	Additional arguments passed to `corrplot`.

Value

Invisibly, the association matrix.

Examples

```
df <- data.frame(
  A = factor(sample(c("Yes", "No"), 50, replace = TRUE)),
  B = factor(sample(c("High", "Low"), 50, replace = TRUE)),
  C = factor(sample(c("Group1", "Group2", "Group3"), 50, replace = TRUE))
)
plot_categorical_associations(df)
```

plot_cox_forest *Forest Plot of Hazard Ratios with Confidence Intervals*

Description

Generates a publication/deck-ready horizontal forest plot of hazard ratios.

Usage

```
plot_cox_forest(
  data,
  label_col = "index_label",
  hr_col = "estimate",
  low_col = "conf.low",
  high_col = "conf.high",
  p_col = NULL,
  color = "#9D2235",
  scale = c("hr", "log_hr"),
  color_by = c("none", "significance"),
  order_by = c("none", "magnitude", "pvalue"),
  x_limits = NULL,
  x_breaks = NULL,
  title = NULL,
  caption = NULL,
  base_size = 13
)
```

Arguments

data	Data frame containing tidy Cox regression results (e.g. from broom::tidy or multi-model runs).
label_col	Column name for labels / terms (character string or unquoted).
hr_col	Column name for hazard ratios, always on the HR (not log) scale (default: "estimate").
low_col	Column name for lower CI bounds, on the HR scale (default: "conf.low").
high_col	Column name for upper CI bounds, on the HR scale (default: "conf.high").
p_col	Optional column name for p-values. Required for color_by = "significance" and order_by = "pvalue".
color	Primary color for points and error bars (default: Temple Cherry "#9D2235").
scale	One of "hr" (default; plots hazard ratios with a reference line at 1) or "log_hr" (log-transforms hr_col/low_col/high_col for plotting, with a reference line at 0).
color_by	One of "none" (default) or "significance", which colors points/error bars by whether p_col is below 0.05 and adds a legend.
order_by	One of "none" (default; original row order), "magnitude" (sorts by abs(log(estimate)) descending), or "pvalue" (sorts by p_col ascending).
x_limits	Optional 2-element numeric vector for x-axis limits.
x_breaks	Optional numeric vector of axis break points.

title	Optional plot title.
caption	Optional plot caption.
base_size	Base font size (default: 13).

Value

A ggplot2 object.

See Also

[plot_cox_forest_multi()]

---

plot_cox_forest_multi *Forest Plot of Hazard Ratios for a Multivariable Cox Model*

Description

Generates a forest plot from a [cbe_cox_multi()] result, grouping terms into per-variable blocks (facets) so factor levels stay visually attached to their parent variable.

Usage

```
plot_cox_forest_multi(
  x,
  color = "#9D2235",
  scale = c("hr", "log_hr"),
  color_by = c("none", "significance"),
  order_by = c("none", "magnitude", "pvalue"),
  x_limits = NULL,
  x_breaks = NULL,
  title = NULL,
  caption = NULL,
  base_size = 13
)
```

Arguments

x	A cbe_cox_multi object from [cbe_cox_multi()].
color	Primary color for points and error bars (default: Temple Cherry "#9D2235").
scale	One of "hr" (default) or "log_hr". See [plot_cox_forest()].
color_by	One of "none" (default) or "significance". See [plot_cox_forest()].
order_by	One of "none" (default), "magnitude", or "pvalue", applied within each variable's block. See [plot_cox_forest()].
x_limits	Optional 2-element numeric vector for x-axis limits.
x_breaks	Optional numeric vector of axis break points.
title	Optional plot title.
caption	Optional plot caption.
base_size	Base font size (default: 13).

Value

A ggplot2 object.

See Also

[cbe_cox_multi()], [plot_cox_forest()]

`plot_cox_marginal` *Marginal Event Probability / Relative Hazard Diagnostic Plot*

Description

Plots a continuous predictor against a model-derived curve, with a 95 for `scale = "prob"`, overlaid actual binary event observations.

Usage

```
plot_cox_marginal(
  fit,
  data,
  feature = NULL,
  status_col = "status",
  scale = c("prob", "hr", "log_hr"),
  color = "#9D2235",
  base_size = 12
)
```

Arguments

`fit`	A fitted `survival::coxph` model or a `cbe_cox` object.
`data`	Data frame containing survival inputs.
`feature`	Character name of continuous predictor column.
`status_col`	Character name of status/event column (0 = censored, 1 = event; default: "status"). Only used when `scale = "prob"`.
`scale`	One of `"prob"` (default; predicted probability of the event), `"hr"` (relative hazard, i.e. `exp()` of the centered linear predictor), or `"log_hr"` (the centered linear predictor itself).
`color`	Primary color for the fitted curve and CI band (default: Temple Cherry `"#9D2235"`).
`base_size`	Base font size (default: 12).

Value

A ggplot2 object.

plot_cox_survival *Model-Predicted Survival Curves Stratified by Predictor*

Description

Generates stratified predicted survival curves from a Cox model. For continuous predictors, values are automatically binned into quantile strata (e.g. quartiles).

Usage

```
plot_cox_survival(
  fit,
  data,
  feature = NULL,
  id_col = NULL,
  n_tiles = 4,
  label_endpoints = TRUE,
  overlay_km = FALSE,
  base_size = 12
)
```

Arguments

fit	A fitted survival::coxph model or a cbe_cox object.
data	Data frame used for fitting the model.
feature	Character name of the predictor column.
id_col	Character name of the subject ID column (default: NULL, which uses the row number).
n_tiles	Number of quantile bins for continuous predictors (default: 4).
label_endpoints	Logical; if TRUE, repels labels for stratum values at the final time point.
overlay_km	Logical (default: FALSE). If TRUE, overlays observed Kaplan-Meier step curves (dashed) for each stratum on top of the Cox-predicted curves (solid), sharing color by stratum, with a linetype legend distinguishing "Cox-predicted" from "KM observed". The event time/status used are those the model was fit with (fit$y).
base_size	Base font size (default: 12).

Value

A ggplot2 object.

plot_dynamic_trajectory *Longitudinal Biomarker Trajectories by Cohort / Outcome*

Description

Plots mean trajectory over time with standard error bars, stratified by outcome or treatment cohort (e.g. Survived vs. Died).

Usage

```
plot_dynamic_trajectory(
  data,
  value_col,
  time_col,
  group_col,
  facet_col = NULL,
  colors = c(Survived = "#6F6F6F", Died = "#9D2235"),
  base_size = 12
)
```

Arguments

data	Long-format data frame containing longitudinal measurements.
value_col	Character name of measurement column.
time_col	Character name of time point column.
group_col	Character name of cohort/grouping column.
facet_col	Optional character name of domain or biomarker column to facet wrap.
colors	Named character vector of colors matching group levels. Defaults to c(Survived = "#6F6F6F", Died = "#9D2235").
base_size	Base font size (default: 12).

Value

A ggplot2 object.

plot_group_comparison *Group Comparison Bar Chart with Standard Errors*

Description

Faceted bar chart comparing biomarker means and standard errors across groups.

Usage

```
plot_group_comparison(
  data,
  value_col,
  group_col,
  facet_col = NULL,
  se_col = NULL,
  colors = c(Survived = "#6F6F6F", Died = "#9D2235"),
  base_size = 13
)
```

Arguments

data	Data frame containing summary means and standard errors (or raw data).
value_col	Column containing index values or means.
group_col	Cohort/outcome grouping column (e.g. "outcome_label").
facet_col	Column to facet wrap across (e.g. "index_label").
se_col	Optional standard error column. If NULL, calculated automatically.
colors	Named color vector (default: c(Survived = "#6F6F6F", Died = "#9D2235")).
base_size	Base font size (default: 13).

Value

A ggplot2 object.

---

plot_missingness *Missing Data Audit Plot*

Description

Generates dodged percentage bar charts comparing data completeness across modalities.

Usage

```
plot_missingness(data, base_size = 13)
```

Arguments

data	Data frame with columns: parameter, pct (fraction missing), method.
base_size	Base font size (default: 13).

Value

A ggplot2 object.

plot_pca_bi *PCA Biplot*

Description

Biplot of feature loading vectors against two principal components.

Usage

```
plot_pca_bi(pca_model, newdata, column, x = 1, y = 2)
```

Arguments

pca_model	A prcomp object.
newdata	Data to project onto pca_model.
column	Column in newdata to use as point labels.
x, y	Which principal components to plot on the x/y axes (default 1, 2).

Value

A ggplot object.

Examples

```
pca_model <- prcomp(mtcars, center = TRUE, scale. = TRUE)
mtcars2 <- tibble::rownames_to_column(mtcars, "model")
plot_pca_bi(pca_model, mtcars2, column = "model")
```

plot_survival_km *Standardized Kaplan-Meier Survival Curve*

Description

Renders a clean Kaplan-Meier curve using the Temple Cherry palette, with censor marks, confidence ribbons, and percentage-formatted survival axis.

Usage

```
plot_survival_km(
  data,
  time_col,
  status_col,
  group_col = NULL,
  color = "#9D2235",
  title = "Kaplan-Meier Survival Estimate",
  caption = NULL,
  base_size = 13
)
```

Arguments

data	Data frame containing survival inputs.
time_col	Character string naming the follow-up time column.
status_col	Character string naming the event status column (1 = event, 0 = censored).
group_col	Optional character string naming a stratification grouping column.
color	Primary line/ribbon color (default: Temple Cherry "#9D2235").
title	Optional plot title.
caption	Optional plot caption.
base_size	Base font size (default: 13).

Value

A ggplot2 object.

---

predict.coxnet_model *Predict From a ‘coxnet‘ Model*

Description

Predict From a ‘coxnet‘ Model

Usage

```
## S3 method for class 'coxnet_model'
predict(
  object,
  new_data,
  type = c("linear_pred", "survival", "time"),
  penalty = NULL,
  eval_time = NULL,
  increasing = TRUE,
  ...
)
```

Arguments

object	A [coxnet()] model.
new_data	A data frame (or matrix) of new predictors.
type	‘"linear_pred"‘ for the linear predictor, ‘"survival"‘ for survival probabilities at ‘eval_time‘, or ‘"time"‘ for the restricted mean survival time: the area under the predicted survival curve from 0 to the last time in the training data.
penalty	The penalty to predict at; defaults to the model’s ‘penalty‘.
eval_time	For ‘type = "survival"‘, the times to predict survival at.
increasing	For ‘type = "linear_pred"‘: if ‘TRUE‘ (the default, as in tidymodels’ **censored** package), the sign is flipped so larger values mean longer survival. Use ‘FALSE‘ for glmnet’s own sign, where larger values mean higher risk.
...	Not used.

Value

A tibble with one row per row of ‘new_data‘: ‘.pred_linear_pred‘, ‘.pred_time‘, or ‘.pred‘, a list-column of tibbles with ‘.eval_time‘ and ‘.pred_survival‘.

Survival is $S(t \mid x) = \exp(-H_0(t)e^{x^\top \beta})$, with H_0 the Breslow estimate of the cumulative baseline hazard from the training data at ‘penalty‘. Each row’s covariates are taken as constant from time 0; to predict along a subject’s start/stop covariate path, and score it, use [cv_coxnet()].

`predict.joint_model` *Predict Method for Joint Models*

Description

Generates multi-paradigm predictions from a fitted `joint_model`: dynamic survival probabilities from the Cox model, binary event probabilities from the status model (both raw and calibrated), and expected duration from the time model.

Usage

```
## S3 method for class 'joint_model'
predict(object, new_data = NULL, eval_time = NULL, type = NULL, ...)
```

Arguments

`object`	A `joint_model` object.
`new_data`	Optional new data frame to predict upon. If `NULL`, predicts on training data.
`eval_time`	Horizon times for survival probabilities. Defaults to the model’s `eval_time`.
`type`	Optional single prediction type, to return only that column as a one-column tibble: `"survival"`, `"time"` (or `"numeric"`), `"prob"` (calibrated status probability), `"status"`, `"linear_pred"`, `"risk_score"`, `"stack_linear_pred"` or `"stack_risk_score"`. `NULL` (the default) or `"all"` returns every column.
`...`	Not used; checked with `check_dots_empty`.

Details

Predictors are re-derived from `new_data` with `forge` against the model’s training blueprint, so factor levels and dummy columns match training exactly, even if `new_data` does not exhibit every level.

The result has one row per row of `new_data`, also for start/stop data, and each row is predicted from its own covariates, taken as constant from time 0. That is not how a subject’s survival is scored along their start/stop covariate path; see `cv_joint_model` (argument `covariates`).

Value

A tibble with columns:

`.pred_survival`	Nested list of survival probability curves over `eval_time`.
`.pred_status`	Predicted event probability from the status model.
`.pred_status_calibrated`	Calibrated event probability (if calibration was enabled).
`.pred_time`	Predicted duration from the time model.

`.pred_linear_pred`
: Linear predictor from the Cox model (higher = longer survival).

`.pred_risk_score`
: Relative hazard risk score from the Cox model.

`.pred_stack_linear_pred`
: Linear predictor from the stacked meta-learner (if `engine = "stacks"`).

`.pred_stack_risk_score`
: Relative hazard risk score from the stacked meta-learner (if `engine = "stacks"`).

`print.cbe_cox` *Print Method for cbe_cox Object*

Description

Print Method for cbe_cox Object

Usage

```
## S3 method for class 'cbe_cox'
print(x, ...)
```

Arguments

`x`	Object of class `cbe_cox`
`...`	Additional arguments (unused)

`print.cbe_cox_check` *Print Method for cbe_cox_check Object*

Description

Print Method for cbe_cox_check Object

Usage

```
## S3 method for class 'cbe_cox_check'
print(x, ...)
```

Arguments

`x`	Object of class `cbe_cox_check`
`...`	Additional arguments (unused)

print.cbe_cox_multi *Print Method for cbe_cox_multi Object*

Description

Print Method for cbe_cox_multi Object

Usage

```
## S3 method for class 'cbe_cox_multi'
print(x, ...)
```

Arguments

x	Object of class cbe_cox_multi
...	Additional arguments (unused)

print.cbe_km *Print Method for cbe_km Object*

Description

Print Method for cbe_km Object

Usage

```
## S3 method for class 'cbe_km'
print(x, ...)
```

Arguments

x	Object of class cbe_km
...	Additional arguments (unused)

print.joint_model *Print Method for Joint Models*

Description

Print Method for Joint Models

Usage

```
## S3 method for class 'joint_model'
print(x, ...)
```

Arguments

x	A joint_model object.
...	Additional arguments.

`print_pipeline` *Print the dependency graph as a PNG image*

Description

Renders via igraph/ggraph with a layered (Sugiyama) DAG layout and canvas sizing adapted to the graph's own node density, rather than the previous Graphviz/DiagrammeR renderer, which laid large graphs out extremely wide and flat (e.g. the full compute graph rendered at 9019x1140px, with text too small to read and the arrows between boxes impossible to follow).

Usage

```
print_pipeline(
  objects,
  path,
  drop_stages = character(0),
  drop_edges_to = character(0),
  title = NULL,
  base_font_size = 9,
  label_wrap_width = 18,
  min_width_in = 8,
  min_height_in = 3,
  max_width_in = 40,
  width_per_node_in = 2.1,
  dpi = 200
)
```

Arguments

`objects`	The full list of FilePath, FileUses, and FileOutputs objects, or an already-built igraph object (see as_igraph()).
`path`	The path where the PNG image will be saved.
`drop_stages`	Character vector of 'stage' values to omit entirely (node and its edges). Use "output_report" on a crowded overview graph to hide the "<X> PDF"/"<X> DOCX" leaf nodes, which add little beyond what their parent QMD node already conveys.
`drop_edges_to`	Character vector of node names whose *incoming* edges should be omitted, while keeping the node itself visible. Use this for a node like an introduction/summary report that every other report is wired to depend on purely to force it to render last, not because of a real analytical dependency – dropping just the edges avoids a dense fan-in that conveys no real relationship.
`title`	Optional title drawn on the plot itself. Leave NULL when the image is embedded with its own Quarto fig-cap, to avoid a duplicate- looking caption.
`base_font_size`	Base font size (pt) for node labels; the title uses 'base_font_size + 4'.
`label_wrap_width`	Character width at which node labels are wrapped.
`min_width_in, max_width_in`	Minimum and maximum plot width, in inches. Width scales with 'width_per_node_in' between these bounds.

min_height_in	Minimum plot height, in inches; height also scales with the number of pipeline layers.
width_per_node_in	Width, in inches, added per node along the graph's horizontal extent, before clamping to 'min_width_in'/'max_width_in'.
dpi	Resolution (dots per inch) of the saved PNG.

---

proc_pca *Process and Plot Principal Component Analysis (PCA)*

Description

Extracts a per-component variance summary from a PCA fit. Accepts either an existing prcomp object or raw data, in which case prcomp is run first.

Usage

```
proc_pca(data, center = TRUE, scale = TRUE, ...)
```

Arguments

data	A 'prcomp' object, or a numeric matrix/data frame to fit PCA on.
center, scale	Passed to prcomp (as 'center' and 'scale.') when 'data' is raw data; ignored when 'data' is already a 'prcomp' object.
...	Further arguments passed to prcomp when 'data' is raw data.

Value

A tibble with one row per component: 'component', 'eigenvalue', 'difference', 'proportion', 'variance_pct', 'cum_variance_pct'.

Examples

```
proc_pca(prcomp(mtcars[, 1:4], scale. = TRUE))

# Or fit the PCA in one step
proc_pca(mtcars[, 1:4], scale = TRUE)
```

---

profvis_summary *Summarize a 'profvis' Profile*

Description

Tabulates the samples of a [profvis::profvis()] profile by function: memory, memory increments, call frequency, stack depth, and memory over time.

Usage

```
profvis_summary(prof)
```

Arguments

prof	A ‘profvis‘ object.

Details

Memory columns sum the sampled ‘memalloc‘/‘meminc‘ values, so they rank functions by how much memory was allocated while they were on the stack; they are not exact allocation totals.

Value

A named list of tibbles:

‘memory_by_function‘ Summed ‘memalloc‘, share, calls, and memory per call, by function.

‘memory_increment_by_function‘ Summed ‘meminc‘, share, calls, and increment per call, by function.

‘calls_by_function‘ Samples per function and share.

‘deepest_calls‘ Each function’s deepest sample, deepest first.

‘memory_over_time‘ Summed ‘memalloc‘ per sample time.

Examples

```
## Not run:
prof <- profvis::profvis(for (i in 1:50) sort(runif(1e5)))
profvis_summary(prof)$memory_by_function

## End(Not run)
```

---

racing_workflows *Racing Methods and Controls for Survival Workflows*

Description

Provides an interface to **finetune** racing methods (tune_race_anova, tune_race_win_loss, and control_race) tailored for Tidymodels survival workflows, workflow sets, and nested cross-validation.

Details

Racing evaluates candidate parameter configurations across resample folds sequentially, eliminating unpromising candidates using ANOVA models or Bradley-Terry win-fraction models before evaluating all folds on all grid points. This significantly accelerates hyperparameter optimization for survival models such as coxnet and multi-model workflow sets.

range_norm *Range Normalization*

Description

Like min_max_norm, but treats every numeric column as coming from a single combined distribution — one global min/max is used to rescale all of them together, rather than normalizing each column independently. Unlike min_max_norm, non-numeric columns are kept (only the numeric ones are transformed) — matching the original implementation this was migrated from.

Usage

```
range_norm(obj)
```

Arguments

obj	A numeric vector, matrix, data frame, or tibble.

Value

A normalized object of the same dimensions and class as obj.

Examples

```
range_norm(data.frame(x = c(1, 2, 3), y = c(-10, 0, 10)))
```

---

read_database_from_excel

Read an R Database from an Excel Workbook

Description

Reads all worksheets from an Excel workbook into a named list of tibbles, optionally extracting and attaching the METADATA sheet as an attribute.

Usage

```
read_database_from_excel(
  path,
  include_metadata = TRUE,
  metadata_sheet = "METADATA",
  ...
)

cbe_read_database(
  path,
  include_metadata = TRUE,
  metadata_sheet = "METADATA",
  ...
)
```

Arguments

path	Path to the .xls/.xlsx file.
include_metadata	Logical; if TRUE (default) and a sheet named metadata_sheet is found, it is separated and attached as attribute "metadata" on the returned database list.
metadata_sheet	Name of the metadata sheet to recognize (default: "METADATA").
...	Additional arguments passed to read_excel.

Value

A named list of tibbles. If include_metadata = TRUE and the metadata sheet was found, it is accessible via attr(result, "metadata").

Examples

```
## Not run:
db <- read_database_from_excel("clinical_database.xlsx")
meta <- attr(db, "metadata")

## End(Not run)
```

---

read_data_manifest *Read Data Manifest File*

Description

Reads a CSV manifest file listing frozen dataset copies and their repo-relative sources. Expected columns: file (filename in destination folder) and source (repo-relative path). Handles UTF-8 BOM headers created by spreadsheet applications like Microsoft Excel.

Usage

```
read_data_manifest(dir = ".", manifest_file = "manifest.csv")
```

Arguments

dir	Directory containing the manifest file (default: current directory).
manifest_file	Name of the manifest CSV file (default: "manifest.csv").

Value

A data frame containing the manifest entries.

`read_excel_multiple_headers` *Read Excel Data With Multi-Row Column Headers*

Description

Variant of `read_excel` for sheets where column names are split across multiple header rows. The rows are concatenated in order, joined with `sep`. Supports hierarchical/merged headers via horizontal forward-filling across upper header tiers, isolates text parsing for headers so data `col_types` do not collide, and performs robust name repair.

Usage

```
read_excel_multiple_headers(
  path,
  n_header_rows = 2L,
  sheet = 1,
  fill_merged = FALSE,
  sep = " | ",
  clean_names = FALSE,
  trim = TRUE,
  ...
)
```

Arguments

`path`	Path to the `.xls`/`.xlsx` file.
`n_header_rows`	Number of header rows in the sheet (positive integer >= 1).
`sheet`	Sheet to read. Either a string (the name of a sheet), or an integer (the position of the sheet). Defaults to `1`.
`fill_merged`	Logical (default `FALSE`); if `TRUE`, handles merged cells in upper header rows by forward-filling non-empty values left-to-right across columns before vertical concatenation.
`sep`	Character string used to separate header levels. Default `" | "`.
`clean_names`	Logical (default `FALSE`); if `TRUE`, applies `clean_names` to sanitize the final column names.
`trim`	Logical (default `TRUE`); whether to trim leading/trailing whitespace from header cells.
`...`	Additional arguments passed to `read_excel` for reading the data body (e.g. `col_types`, `na`, `guess_max`).

Value

A tibble.

Examples

```
## Not run:
read_excel_multiple_headers("workbook.xlsx", n_header_rows = 2)
read_excel_multiple_headers("workbook.xlsx", n_header_rows = 2, fill_merged = TRUE)
read_excel_multiple_headers("workbook.xlsx", n_header_rows = 2, clean_names = TRUE)

## End(Not run)
```

---

read_mapped_section_data

Read One Section's Data Using a Column Mapping

Description

Generic schema-enforced reader: reads whichever file corresponds to an `INDEX` value in a mapping table, renames raw headers to standardized names, resolves duplicate columns per `duplicate_action`, and tolerates missing or unmapped columns.

Usage

```
read_mapped_section_data(
  mapping,
  index,
  file = NULL,
  data_dir = NULL,
  sheet = NULL,
  header_rows = 1
)
```

Arguments

`mapping`	Validated column mapping table.
`index`	Character string identifying the section/domain in `mapping$INDEX`.
`file`	Optional explicit path to the data file.
`data_dir`	Directory to search for the section's file if `file` is NULL.
`sheet`	Sheet name or number for Excel files.
`header_rows`	Number of header rows to concatenate (default: 1).

Value

A tibble with standardized column names.

read_raw_table *Read Raw Table with Multi-Row Header Support*

Description

Read Raw Table with Multi-Row Header Support

Usage

```
read_raw_table(file, sheet = NULL, header_rows = 1)
```

Arguments

file	Path to .csv or .xlsx file.
sheet	Sheet name or number (for Excel).
header_rows	Number of rows to combine into column headers (default: 1).

Value

A tibble with raw data.

read_search *Search a Directory Tree for File-Read Calls*

Description

Convenience wrapper around find_code that searches for common data-import function calls (readRDS, read.csv, read_excel, read_workbook, read_excel_multiple_headers, etc.) across a directory tree.

Usage

```
read_search(path, include_comments = TRUE, workers = NULL)
```

Arguments

path	Character. Root directory to search.
include_comments	Logical. Passed through to find_code. Defaults to TRUE.
workers	Integer number of parallel workers to use when **future/furrr** are installed. Defaults to NULL, which caps at available cores minus one. Falls back to sequential search when **future/furrr** are not installed.

Value

A tibble of matches (see find_code) with an additional pattern column recording which read function matched.

Examples

```
## Not run:
read_search(here::here("analysis"))

## End(Not run)
```

`read_workbook` *Read Every Sheet of an Excel Workbook*

Description

Variant of `read_excel` that reads every sheet in a workbook into a named list of tibbles, named after each sheet.

Usage

```
read_workbook(path, ...)
```

Arguments

`path`	Path to the `.xls`/`.xlsx` file.
`...`	Additional arguments passed to `read_excel`.

Value

A named list of tibbles, one per sheet.

Examples

```
## Not run:
read_workbook("workbook.xlsx")

## End(Not run)
```

`render` *Render Quarto and R Markdown Documents to Multiple Formats With Timing*

Description

Renders one or more Quarto (`.qmd`) or R Markdown (`.Rmd`) documents to multiple output formats (by default both PDF and DOCX, or formats defined in YAML or compute graphs) sequentially or in parallel, recording execution duration for each document/format combination. Useful for workflows where reviewers need a Word (`.docx`) file for track changes/edits while publication and archival copies need PDF.

Usage

```
render(
  path,
  formats = getOption("pipeline.default_formats", c("pdf", "docx")),
  pattern = "\\.(qmd|Rmd|rmd)$",
  engine = c("auto", "quarto", "rmarkdown"),
  workers = NULL,
  ...
)

render_me(
  path,
  formats = getOption("pipeline.default_formats", c("pdf", "docx")),
  pattern = "\\.(qmd|Rmd|rmd)$",
  engine = c("auto", "quarto", "rmarkdown"),
  workers = NULL,
  ...
)
```

Arguments

path	Target document(s) to render. Can be: • A character string specifying a file path or directory. • A character vector of file paths. • A list of file paths (e.g. `list("report1.qmd", "report2.Rmd")`). • An S4 compute graph object (`FileOutputs` or `FilePath`). • A list of compute graph objects, such as a topological render plan returned by `get_render_plan` or a pipeline list from `pipeline_config`. • A named list or list of lists specifying per-document paths and formats (e.g. `list("report1.qmd" = c("pdf", "docx"), "report2.Rmd" = "html")`).
formats	Output format(s) to generate. Defaults to `getOption("pipeline.default_formats", c("pdf", "docx"))`. Can be: • A character vector of formats (e.g. `c("pdf", "docx")`, `"html"`, `"gfm"`). • `"yaml"` or `"auto"` (or `NULL`) to read and render the output formats declared in each document's own YAML frontmatter. • Standard shorthand formats (`"pdf"`, `"docx"`, `"html"`, `"gfm"`) are automatically translated to R Markdown equivalents (`"pdf_document"`, `"word_document"`, `"html_document"`, `"github_document"`) when rendering with **rmarkdown**. When a document in a compute graph defines its own deliverable output formats (via `create_qmd_renderer`), those document-level formats override the default `formats`.
pattern	Optional regular expression to filter files when `path` is a directory. Defaults to `"\.(qmd|Rmd|rmd)$"`.
engine	Rendering engine: `"auto"` (default), `"quarto"`, or `"rmarkdown"`. When `"auto"`, `.qmd` files are rendered via **quarto**, and `.Rmd` files are rendered via **quarto** if available or **rmarkdown** otherwise.
workers	Integer specifying the number of multisession parallel workers when rendering multiple documents. Defaults to `NULL`, which caps at available cores minus one. Ignored (rendering falls back to sequential) when the **future/furrr** packages are not installed.

...	Additional options passed directly to the rendering backend (quarto::quarto_render or rmarkdown::render), such as params, execute_params, output_dir, clean, or quiet.

Value

A data frame containing:

file	Base filename of the rendered document
output	Output format (e.g., "pdf", "docx")
duration	Execution time in seconds
status	"success" or error condition message

Examples

```
## Not run:
# Render a single Quarto report to both PDF and DOCX
render(here::here("analysis", "report.qmd"))

# Render a single R Markdown report to both PDF and Word
render(here::here("analysis", "report.Rmd"))

# Render using formats declared in document's own YAML
render("analysis/report.qmd", formats = "yaml")

# Render a compute graph render plan
# plan <- get_render_plan(pipeline)
# render(plan)

# Render all QMD and Rmd reports in a directory in parallel
render("analysis/")

## End(Not run)
```

---

review_config *Get or set global multi-reviewer review-tracking configuration*

Description

Decouples project-specific reviewer identities from the review-tracking mechanics in cbe_docx_review_extract, mirroring how pipeline_config decouples study name and stage labels from the core compute-graph classes: the number and names of reviewers are data supplied by the calling project, not hardcoded here.

Usage

```
review_config(
  fork_reviewers = NULL,
  signoff_reviewers = NULL,
  documents_sheet_mode = NULL,
  csv_reviewer_columns = NULL,
  pipeline_catalog = NULL,
```

```
  stem_aliases = NULL,
  analysis_dir = NULL,
  input_dirs = NULL,
  reset = FALSE
)
```

Arguments

fork_reviewers	Character vector of fork-reviewer identifiers. An id is used as a column name, as a fork-workbook file suffix and (with documents_sheet_mode = "per_reviewer") in a sheet name, so it is validated: ids must be unique (ignoring case) and consist of 1-20 letters, digits or underscores; they must not be the name of a built-in tracker column (for example file, author or resolved) and must not end in _comment. A violation is an error, and nothing is changed.
signoff_reviewers	Character vector of document-level sign-off reviewer identifiers (same rules as fork_reviewers; an id cannot be both a fork and a sign-off reviewer). Defaults to none.
documents_sheet_mode	One of "columns" (default; a single "Documents" sheet with one column, or column pair, per reviewer) or "per_reviewer" (a shared "Documents" sheet with counts and resolved only, plus one additional "Documents_<id>" sheet per reviewer carrying just that reviewer's status/comment columns). This applies to the master workbook; a fork workbook always has one plain "Documents" sheet.
csv_reviewer_columns	Which reviewer columns the CSV exports (comments.csv, tracked_changes.csv, suggested_changes.csv) contain. "fork" (default) keeps the fork reviewers' sign-off and comment columns but leaves out the sign-off reviewers' columns and the Word author column; "none" leaves out every reviewer column and the author; "all" writes every column, including the sign-off reviewers' and the author, which is what the CSVs held before this option existed.
pipeline_catalog	The analysis stages the reviewed documents belong to, used to label (pipeline_stage) and order documents. A list of stages, each a list with a stem (the document's file name without extension, date and reviewer initials, compared in lower case) and optionally heading, name and stage (the label shown; default "Heading <heading>: <name>", else the name, else the stem); a data frame with those columns also works. The order of the list is the order of the documents. The package ships *no* stages: unless a catalog (or a reports_to_render.xlsx manifest in analysis_dir) is supplied, every document gets the stage "Unknown / Extra". A file whose stem is not in the catalog is matched to the most similar stem when they are at least 60 percent alike. Use list() to clear.
stem_aliases	Named character vector of file-name stems (as typed, for example a recurring typo) and the catalog stems they stand for. Defaults to none.
analysis_dir	Folder, relative to the working directory (or to the TEMPLECBE_ANALYSIS_ROOT environment variable) or absolute, that holds reports_to_render.xlsx and the .qmd sources shown on the docXwalk sheet. Defaults to none: no manifest or source index is looked up. Use character(0) to clear.
input_dirs	Character vector of folders (relative to the working directory or TEMPLECBE_ANALYSIS_ROOT, or absolute) in which cbe_docx_review_extract looks, in order, for .docx

files when input_dir is not given. Defaults to none, so the working directory is used. Use character(0) to clear.

reset — If TRUE, forget everything set earlier (every argument above) and return to the defaults before applying any other argument given in the same call; the arguments are validated first, so a rejected call changes nothing. review_config(reset = TRUE) alone just restores the defaults. Defaults to FALSE.

Details

Two reviewer categories are supported:

- fork_reviewers: reviewers who sign off on individual comments and suggested changes. Each gets a personal fork workbook (review_tracker_<id>.xlsx) that holds only that reviewer's own columns: no other reviewer's columns, no sign-off reviewer's columns and no Word author names appear on any sheet, and the "Documents" sheet carries just the counts and that reviewer's own rolled-up column (there is no "Documents_<id>" sheet in a fork). In a fork the resolved columns reflect that reviewer's own sign-offs only; in the master workbook a document's resolved status is TRUE only when every fork reviewer has signed off on every comment and suggested change in that document.
- signoff_reviewers: reviewers who sign off once per document (e.g. a final PI read), tracked on the master workbook's "Documents" sheet (a status column plus a free-text comment column). They are not counted toward resolved, and they never appear in a fork workbook or, by default, in the CSV exports (see csv_reviewer_columns). The master's Comments and SuggestedChanges sheets also carry their column pair, so that anything typed there is kept.

Existing review_tracker_*.xlsx files continue to work: column data is preserved by header union regardless of configuration, but for the app's reviewer-aware behavior (rollups, resolved status, fork filtering) to line up with a file's existing columns, keep configuring the same reviewer identifiers used when that file was created.

Value

A list containing the current configuration options.

---

rossi_data *Rossi Recidivism Dataset*

Description

Loads the classic Rossi recidivism study dataset used in biostatistical benchmarks for survival models with time-varying coefficients and covariates (e.g., JSS Vol 61, Code 01). The study tracks 432 convicts released from Maryland state prisons over a 52-week follow-up period.

Usage

```
rossi_data(format = c("csv", "sas"))
```

Arguments

format — Either ‘"csv"‘ (default, reading ‘inst/extdata/rossi.csv‘) or ‘"sas"‘ (reading the native ‘inst/extdata/recid.sas7bdat‘ via [haven::read_sas()]).

Value

A tibble with 432 rows containing recidivism follow-up data:

week Follow-up time until arrest or censoring (1–52 weeks).

arrest Event indicator (1 = arrested, 0 = censored/did not re-offend).

fin Financial aid treatment (1 = received financial aid, 0 = control).

age Age at release from prison in years.

race Race indicator (1 = Black, 0 = other).

wexp Prior full-time work experience (1 = yes, 0 = no).

mar Marital status (1 = married, 0 = unmarried).

paro Release on parole (1 = yes, 0 = no).

prio Number of prior convictions.

educ Education level coded as 2 (grades 2-5) through 6 (some college).

emp1–emp52 Weekly employment status indicator across the 52 weeks.

Source

Rossi, P. H., Berk, R. A., & Lenihan, K. J. (1980). *Money, work, and crime: Some experimental results*. Academic Press.

See Also

[cbe_sas_macro_path()], [run_sas_script()]

Examples

```
df <- rossi_data()
head(df[, 1:10])
```

---

rotation_matrix *PCA Rotation Matrix (Loadings)*

Description

PCA Rotation Matrix (Loadings)

Usage

```
rotation_matrix(PC_mod)

pca_loadings(PC_mod)
```

Arguments

PC_mod	A prcomp object.

Value

A tibble of feature loadings onto each principal component, with a feature_num column ("f1", "f2", ...) for compact labeling in pca_eqns.

Examples

```
pca_model <- prcomp(mtcars, center = TRUE, scale. = TRUE)
rotation_matrix(pca_model)
```

---

run_sas_script *Run a SAS Program in Batch Mode*

Description

Runs a '.sas' file with the SAS executable, writing its log and listing to separate folders next to the program (created if needed).

Usage

```
run_sas_script(
  path,
  sas_path = find_sas(),
  log_dir = file.path(dirname(path), "logs"),
  list_dir = file.path(dirname(path), "list"),
  timeout = 0
)
```

Arguments

path	Path to the '.sas' program.
sas_path	Path to the SAS executable. Defaults to [find_sas()].
log_dir	Folder for the '.log' file. Defaults to 'logs/' beside 'path'.
list_dir	Folder for the '.lst' listing. Defaults to 'list/' beside 'path'.
timeout	Seconds SAS may run before it is stopped, or '0' (the default) for no limit. A run that is stopped is an error that names the log, which holds what SAS had written by then. Only the SAS process that was started is stopped.

Value

The exit status of the SAS process (0 on success, 1 for warnings, 2 for errors). A run that is stopped by 'timeout' is an error.

See Also

[find_sas()], [cbe_sas_macro_path()]

Examples

```
## Not run:
programs <- list.files("validation", pattern = "\\.sas$", full.names = TRUE)
vapply(programs, run_sas_script, integer(1))

## End(Not run)
```

R_names *Clean Column Names, Preserving Originals as Labels*

Description

Runs clean_names on df and stores each column's original name as a **labelled** variable label, so the human-readable original isn't lost when the column name itself becomes machine-friendly.

Usage

```
R_names(df)
```

Arguments

df	A data frame or tibble.

Value

A tibble with cleaned column names and the original names stored as variable labels (see var_label).

Examples

```
df <- tibble::tibble(
  `name with spaces` = 1:10,
  `special * characters` = LETTERS[1:10]
)
R_names(df)
```

scale_color_cbe *Discrete Color Scale for Temple CBE*

Description

Discrete Color Scale for Temple CBE

Usage

```
scale_color_cbe(...)
```

Arguments

...	Arguments passed to ggplot2::scale_color_manual

`scale_colour_temple` *Temple Color and Fill Scales for ggplot2*

Description

Temple Color and Fill Scales for ggplot2

Usage

```
scale_colour_temple(
  palette = c("main", "diverging", "sequential"),
  discrete = NULL,
  reverse = FALSE,
  midpoint = 0,
  ...
)

scale_color_temple(
  palette = c("main", "diverging", "sequential"),
  discrete = NULL,
  reverse = FALSE,
  midpoint = 0,
  ...
)

scale_fill_temple(
  palette = c("main", "diverging", "sequential"),
  discrete = NULL,
  reverse = FALSE,
  midpoint = 0,
  ...
)
```

Arguments

`palette`	One of `"main"`, `"diverging"`, or `"sequential"`.
`discrete`	Logical; a discrete scale (default for `"main"`) or a continuous gradient.
`reverse`	Logical; reverse the color order.
`midpoint`	For continuous `"diverging"` scales, the data value mapped to white (default 0).
`...`	Passed to `discrete_scale`, `scale_colour_gradient2`, or `scale_colour_gradientn`.

Value

A ggplot2 scale.

Examples

```
library(ggplot2)
ggplot(mtcars, aes(wt, mpg, colour = factor(cyl))) +
  geom_point() +
```

```
  scale_colour_temple()

ggplot(faithfuld, aes(waiting, eruptions, fill = density)) +
  geom_raster() +
  scale_fill_temple("sequential", discrete = FALSE)
```

`scale_fill_cbe` *Discrete Fill Scale for Temple CBE*

Description

Discrete Fill Scale for Temple CBE

Usage

```
scale_fill_cbe(...)
```

Arguments

`...`	Arguments passed to `ggplot2::scale_fill_manual`

`scan_data_io` *Audit Data File Read/Write Calls Against a Project's Files on Disk*

Description

Scans `code_path` for read and write calls (via `read_search`/`write_search`), resolves the file paths those calls reference — including `here::here()` calls, and a heuristic fallback for indirect references like `excel_file_paths[["ABG"]]` matched against filenames actually present on disk — and cross-references them against every matching file under `project_root`. The result classifies each such file as a write output the code produces, a read input the code consumes, both, or an orphan with no code reference found (`"unknown"`) — useful for finding stale, missing, or undocumented data deliverables in a project.

Usage

```
scan_data_io(
  code_path,
  project_root = code_path,
  ext = "xlsx",
  strong_read_patterns = c("read_workbook", "read_excel_multiple_headers"),
  include_comments_write = FALSE,
  max_depth = Inf
)
```

Arguments

code_path	Directory of scripts (.R/.Rmd/.qmd) to search for read/write calls.
project_root	Root directory to inventory files under, and to resolve relative/here::here() paths against (useful when auditing a checkout at a different location than the one the scripts were written against). Defaults to code_path.
ext	File extension to audit, without the leading dot (e.g. "xlsx", "csv", "rds"). Defaults to "xlsx".
strong_read_patterns	Character vector of read_search pattern names that should always count as a read of a file with this extension, even on a line that doesn't otherwise mention ext (e.g. a call to a project-specific reader function). Defaults to c("read_workbook", "read_excel_multiple_headers").
include_comments_write	Logical, passed through to write_search. Defaults to FALSE.
max_depth	Maximum folder depth below project_root to inventory files in (default Inf, no limit).

Value

A list with:

writes	Resolved file-write call sites.
inputs	Resolved file-read call sites.
missing_write_dirs	Directories containing a write target that doesn't exist on disk yet, and any matching-extension files present there now — a likely spot for a not-yet-(re)generated deliverable.
files	Every file with extension ext under project_root, tagged write_output/workflow_input/unknown and by which script(s) reference it.

Examples

```
## Not run:
scan_data_io(here::here("analysis"), ext = "xlsx")

## End(Not run)
```

---

sd.error *Compute standard error*

Description

Compute standard error

Usage

```
sd.error(x, na.rm = FALSE)
```

Arguments

x	vector of data
na.rm	should NA values be removed, TRUE or FALSE, default is FALSE

Value

numeric value representing standard error

Examples

```
sd.error(mtcars$mpg)
```

---

significance_stars *P-value Significance Stars*

Description

P-value Significance Stars

Usage

```
significance_stars(p_value)
```

Arguments

p_value	A numeric vector of p-values.

Value

A character vector of significance stars.

Examples

```
significance_stars(c(0.0001, 0.02, 0.2, 0.8))
```

---

simulate_section_data *Simulate Synthetic Section Data From a Schema Mapping*

Description

Generates synthetic data honoring the column roles declared in a validated mapping table. If the pslongSim package is available, it is leveraged for baseline subject scaffolding; otherwise, an internal robust generator produces demographic and longitudinal grids.

Usage

```
simulate_section_data(
  mapping,
  index,
  n_subjects = 20,
  seed = 1,
  missing_rate = 0,
  time_levels = NULL
)
```

Arguments

mapping	Validated column mapping table; see validate_column_mapping.
index	Character string identifying the section/domain in mapping$INDEX to simulate.
n_subjects	Number of synthetic subjects to generate (default: 20).
seed	Random seed for reproducibility (default: 1).
missing_rate	Probability in [0, 1] that any non-ID, non-time cell is set to NA.
time_levels	Optional character vector of levels for the section's Time_var column (default: c("T0", "T1", "T2", "T3")).

Value

A tibble shaped according to the standardized new columns for the specified section.

---

single_t_test *Single T-Test, Tidied*

Description

Runs a t-test comparing .var between the two levels of .class, tidied into one row with group sizes/SDs and fold-change added.

Usage

```
single_t_test(
  .data,
  .var,
  .class,
  .id = NULL,
  alternative = "two.sided",
  conf.level = 0.95,
  paired = FALSE,
  ...
)
```

Arguments

.data	A data frame or tibble.
.var	Name (string) of the continuous column to test.
.class	Name (string) of a binary (2-level) classification column.
.id	Name (string, optional) of a subject/record identifier column. When paired = TRUE, pass this to pair observations by matching .id across the two groups rather than by row position — every id must have exactly one observation in each group. If paired = TRUE and .id is omitted, observations are paired by row order within each group, which silently produces meaningless results if the two groups aren't already sorted into corresponding order.
alternative	One of "two.sided" (default), "greater", or "less".
conf.level	Confidence level for the interval.
paired	Logical; paired t-test.
...	Additional arguments passed to t.test.

Value

A one-row tibble.

Examples

```
mtcars |>
  dplyr::mutate(am = factor(am)) |>
  single_t_test("mpg", "am")
```

---

step_famd *Factor Analysis of Mixed Data (FAMD) Recipe Step*

Description

'step_famd()' creates a *specification* of a recipe step that converts a mix of numeric and categorical variables into principal components with Factor Analysis of Mixed Data ([FactoMineR::FAMD()]). It is the mixed-data counterpart of [recipes::step_pca()]: numeric variables are standardized and categorical variables are weighted, so each variable contributes on a comparable scale.

Usage

```
step_famd(
  recipe,
  ...,
  role = "predictor",
  trained = FALSE,
  num_comp = 2,
  threshold = NA,
  options = list(),
  res = NULL,
  columns = NULL,
  levels = NULL,
  prefix = "FAMD",
```

```
  keep_original_cols = FALSE,
  skip = FALSE,
  id = recipes::rand_id("famd")
)

## S3 method for class 'step_famd'
tidy(x, type = "coef", ...)
```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose variables for this step. See [recipes::selections()].
role	Role of the new component columns (default ‘"predictor"‘).
trained	A logical indicating whether the step has been trained.
num_comp	Number of components to keep (default 2); ignored when ‘threshold‘ is set. ‘0‘ leaves the data unchanged.
threshold	Fraction of total variance, in (0, 1], that the kept components should cover. Overrides ‘num_comp‘ when not ‘NA‘.
options	A named list of further arguments to [FactoMineR::FAMD()].
res	The fitted FAMD object, once trained.
columns	The selected column names, once trained.
levels	The factor levels of the categorical columns, once trained.
prefix	Prefix of the new column names (default ‘"FAMD"‘), numbered by [recipes::names0()]. It differs from [recipes::step_pca()]’s ‘"PC"‘, so both steps can be used in one recipe.
keep_original_cols	A logical to keep the original variables in the output. Defaults to ‘FALSE‘.
skip	A logical. Should the step be skipped when the recipe is baked by [recipes::bake()]?
id	A character string that is unique to this step.
x	A ‘step_famd‘ object.
type	For ‘tidy()‘: ‘"coef"‘ (variable contributions) or ‘"variance"‘.

Details

The selected variables must include at least one numeric and one categorical (factor, character, or logical) variable; use [recipes::step_pca()] for all-numeric data. Character and logical variables are treated as factors, with levels learned by ‘prep()‘. Variables of any other type, such as dates, are an error in ‘prep()‘: convert them first, e.g. with [recipes::step_date()] or [recipes::step_mutate()]. A variable that was numeric in training must be numeric in the data given to ‘bake()‘. Missing values, and categories that ‘bake()‘ meets but training did not, are errors: impute or collapse levels first, e.g. with [recipes::step_impute_mode()] or [recipes::step_other()].

FAMD has at most $\min(n-1, p+\sum_j(L_j-1))$ dimensions, for n rows, p numeric variables, and categorical variables with L_j levels, which can be more than the number of variables. ‘num_comp‘ is capped there. With ‘threshold‘, the step keeps the fewest components whose cumulative share of variance reaches ‘threshold‘, choosing among all dimensions.

Frequency weights ([hardhat::frequency_weights()]) are passed to FAMD as row weights. Like other unsupervised recipe steps, such as [recipes::step_pca()], it ignores importance weights.

Tidying

‘tidy()‘ with ‘type = "coef"‘ returns each variable’s percentage contribution to each component (‘value‘): FactoMineR’s per-variable summary, which is comparable between numeric and categorical variables. With ‘type = "variance"‘, it returns each component’s variance, cumulative variance, percent variance, and cumulative percent variance, as for [recipes::step_pca()].

Value

An updated recipe object.

References

Pagès J (2004). Analyse factorielle de données mixtes. *Revue de Statistique Appliquée*, 52(4), 93-111.

Examples

```
if (requireNamespace("FactoMineR", quietly = TRUE)) {
  library(recipes)
  rec <- recipe(~ ., data = iris) |>
    step_famd(all_predictors(), num_comp = 3)
  prepped <- prep(rec)
  head(bake(prepped, new_data = NULL))
  tidy(prepped, number = 1, type = "variance")
}
```

`step_lencode_coxnet` *Supervised Linear Encoding of Factors via Penalized Cox Models*

Description

`step_lencode_coxnet` creates a *specification* of a recipe step that encodes categorical predictors into a single numeric variable representing the estimated log hazard ratio from a penalized Cox proportional hazards model on a survival outcome (`Surv(time, status)` or `Surv(tstart, tstop, status)`). Inspired by `embed::step_lencode_glm`. Each factor is encoded on its own: a penalized Cox model with one indicator per non-reference level is fit to the training data at the single `penalty`, the reference level (the first level) is encoded as 0, and levels that `bake()` has not seen, or missing values, are encoded as 0. If a model can’t be fit, a warning is raised and the factor’s levels are all encoded as 0.

Usage

```
step_lencode_coxnet(
  recipe,
  ...,
  outcome = c("time", "status"),
  role = "predictor",
  trained = FALSE,
  penalty = 0.05,
  mixture = 1,
  mapping = NULL,
  skip = FALSE,
```

```
  id = recipes::rand_id("lencode_coxnet")
)

## S3 method for class 'step_lencode_coxnet'
tunable(x, ...)

## S3 method for class 'step_lencode_coxnet'
required_pkgs(x, ...)
```

Arguments

recipe	A recipe object.
...	One or more selector functions to choose variables to be encoded. Must select factor or character columns.
outcome	Character vector naming the survival outcome columns in the training data: `c("time", "status")` or `c("tstart", "tstop", "status")`.
role	For model terms created by this step, what analysis role should they be assigned? By default, the function assumes that the new variables will be used as predictors in a model.
trained	A logical indicating whether the recipe has been trained.
penalty	A non-negative numeric value specifying the L1/L2 penalty for `coxnet`. Default is 0.05.
mixture	Elastic net mixing parameter between 0 and 1. Default is 1 (Lasso).
mapping	A named list of numeric vectors containing the level-to-score mappings, generated during `prep()`.
skip	A logical indicating whether the step should be skipped when the recipe is baked.
id	A unique identifier for the step.
x	A `step_lencode_coxnet` object.

Value

An updated version of `recipe` with the new step added.

Missing values

A missing value in a factor counts as the reference level: in `prep()` it is recoded to the first level before the model is fit, so its row stays in the fit (and, with the other rows of that level, sets the baseline that the other levels are compared with), and in `bake()` it is encoded as 0. `step_lencode_joint_model` follows the same policy.

The training data are encoded with their own fit

`prep()` fits the encoding on the training rows, and `bake(new_data = NULL)`, or baking the training data, encodes those same rows with the coefficients fitted to them: the encoding is *not* out-of-fold (cross-fitted). The encoded training column therefore carries the outcome, so it looks more predictive in the training data than it is for new subjects, and a model fit to it over-trusts it. The effect grows with the number of levels relative to the number of events, so it is worst for factors with many or rare levels, whose coefficients are fitted to a handful of subjects each: a factor with 100 levels and no relation to the outcome can show a clearly better than chance concordance in the

baked training data and chance level on new data. To limit it, use a large `penalty` (most levels then shrink to 0) or pool rare levels before encoding (for example with `recipes::step_other()`). Do not evaluate a model on data that it was encoded with: estimate performance on subjects the step did not see in `prep()`, for example by resampling the whole recipe, so that every `prep()` sees only the analysis set, and not by encoding once and splitting afterwards.

Examples

```
## Not run:
if (requireNamespace("recipes", quietly = TRUE) && requireNamespace("survival", quietly = TRUE)) {
  lung <- survival::lung
  lung_df <- na.omit(lung[, c("time", "status", "sex", "ph.ecog")])
  lung_df$ph.ecog <- factor(lung_df$ph.ecog)
  lung_df$sex <- factor(lung_df$sex)

  rec <- recipes::recipe(time + status ~ ., data = lung_df) |>
    step_lencode_coxnet(ph.ecog, sex, outcome = c("time", "status"))
  prepped <- recipes::prep(rec)
  baked <- recipes::bake(prepped, new_data = NULL)
}

## End(Not run)
```

---

step_lencode_joint_model

Supervised Linear Encoding of Factors via the Cox Component of a Joint Model

---

Description

`step_lencode_joint_model` creates a *specification* of a recipe step that encodes categorical predictors into a numeric score: the log hazard ratio of each level relative to the reference level (the first level, encoded as 0), taken from the Cox component of a `joint_model` fit to that factor alone. Only the Cox component's `.pred_risk_score` is used, so the encoding is a penalized Cox log hazard ratio, like that of `step_lencode_coxnet`; it is *not* a composite that blends the survival, status and follow-up time components of the joint model, and `engine` does not affect it. Levels that `bake()` has not seen, or missing values, are encoded as 0. If a model can't be fit, a warning is raised and the factor's levels are all encoded as 0.

Usage

```
step_lencode_joint_model(
  recipe,
  ...,
  outcome = c("time", "status"),
  role = "predictor",
  trained = FALSE,
  engine = "glmnet",
  penalty = 0.05,
  mapping = NULL,
  skip = FALSE,
  id = recipes::rand_id("lencode_joint_model")
```

```
)

## S3 method for class 'step_lencode_joint_model'
required_pkgs(x, ...)
```

Arguments

recipe	A recipe object.
...	One or more selector functions to choose variables to be encoded.
outcome	Character vector naming the survival outcome columns in the training data, as for step_lencode_coxnet.
role	Role for the encoded variables. Default is "predictor".
trained	A logical indicating whether the step has been trained.
engine	Has no effect on the encoding and is ignored when the step is prepped: only the Cox component of the joint model is used, and it does not depend on the engine. Any value other than the default "glmnet" gives a warning. Kept so that existing code keeps running.
penalty	Penalty of the Cox component. Default is 0.05.
mapping	A named list of level-to-score mappings generated during prep().
skip	Logical; skip step when baking? Default is FALSE.
id	A unique identifier for the step.
x	A step_lencode_joint_model object.

Value

An updated version of recipe.

Missing values

A missing value in a factor counts as the reference level, exactly as in step_lencode_coxnet: in prep() it is recoded to the first level before the model is fit, so its row stays in the fit, and in bake() it is encoded as 0.

The training data are encoded with their own fit

prep() fits the encoding on the training rows, and bake(new_data = NULL), or baking the training data, encodes those same rows with the coefficients fitted to them: the encoding is *not* out-of-fold (cross-fitted). The encoded training column therefore carries the outcome, so it looks more predictive in the training data than it is for new subjects, and a model fit to it over-trusts it. The effect grows with the number of levels relative to the number of events, so it is worst for factors with many or rare levels, whose coefficients are fitted to a handful of subjects each: a factor with 100 levels and no relation to the outcome can show a clearly better than chance concordance in the baked training data and chance level on new data. To limit it, use a large penalty (most levels then shrink to 0) or pool rare levels before encoding (for example with recipes::step_other()). Do not evaluate a model on data that it was encoded with: estimate performance on subjects the step did not see in prep(), for example by resampling the whole recipe, so that every prep() sees only the analysis set, and not by encoding once and splitting afterwards.

stop_if_invalid_manifest

Stop Execution if Data Copies are Invalid or Stale

Description

Halts report execution with an actionable error message if any snapshot copy does not match its upstream source.

Usage

```
stop_if_invalid_manifest(validation, dir = ".")
```

Arguments

validation	Validation tibble returned by validate_data_manifest.
dir	Destination directory name for reporting.

Value

Invisibly returns validation if all files are valid.

summarize_section_by_time

Summarize One Section By Its Time Variable

Description

Generates summary tables (using **gtsummary** by default, or optionally **arsenal**) grouped by whichever column is flagged with Time_var == TRUE in the mapping dictionary.

Usage

```
summarize_section_by_time(
  df,
  mapping,
  index,
  id_cols = NULL,
  engine = c("gtsummary", "arsenal")
)
```

Arguments

df	Data frame already renamed via read_mapped_section_data.
mapping	Validated column mapping table.
index	Character string identifying the section/domain.
id_cols	Column names to exclude from summary (defaults to columns with ID_var == TRUE).
engine	Character string specifying the summary engine: "gtsummary" (the default) or "arsenal".

Value

A tbl_summary object (when engine = "gtsummary") or an arsenal::tableby summary object (when engine = "arsenal").

---

summarize_tune_results *Tune Over 'alpha' for Every Split of a Resample*

Description

Calls [tune_over_alpha()] on each split of 'object' and row-binds the successful fits. Fits that errored are dropped. A split draws its own random 'alpha' values, reproducibly: the map runs with 'furrr_options(seed = TRUE)'.

Usage

```
summarize_tune_results(
  object,
  ...,
  num_alpha_values = 10,
  num_fixed = 6,
  alphas = NULL,
  formulas = NULL,
  progress = FALSE
)
```

Arguments

object	A resampling object with a 'splits' column (e.g. one element of the 'inner_resamples' column of [rsample::nested_cv()]).
...	Arguments passed to [glmnet_IBS()] ('recipe', 'feature_names', 'time_data', 'id_col', 'cox.ties', ...).
num_alpha_values	Total number of 'alpha' values in the grid.
num_fixed	Number of evenly spaced values from 0 to 1 in the grid.
alphas	Optional numeric vector of 'alpha' values to use instead of the generated grid; in formula mode, one per formula.
formulas	Optional character vector of '+'-separated feature sets, or a list of one-sided formulas, each fit as a separate model (see [glmnet_IBS()]'s 'formula').
progress	Show a progress bar.

Value

A tibble of [glmnet_IBS()] results with an 'inner_resamples_splits' column giving the split's position in 'object$splits'.

See Also

[glmnet_IBS()], [tune_over_alpha()], [nested_cv_coxnet()]

summary.tbl_graph *S3 summary method for tbl_graph objects*

Description

S3 summary method for tbl_graph objects

Usage

```
## S3 method for class 'tbl_graph'
summary(object, ...)
```

Arguments

object	A tbl_graph object.
...	Additional arguments (unused; for S3 method consistency).

Value

A tibble of the stage breakdown if ‘object‘ is a pipeline graph (see pipeline_summary), otherwise the default summary() for the next applicable class.

SumNa *Count Total Missing (NA) Values*

Description

Calculates the total number of missing values across a vector, matrix, or data frame, optionally treating additional values (e.g. "NA" as a literal string, or "") as missing too.

Usage

```
SumNa(x, na_list = NULL)
```

Arguments

x	A vector, matrix, or data frame.
na_list	Optional vector of additional values to treat as missing, beyond actual NA.

Value

An integer representing the total count of missing values.

Examples

```
SumNa(c(1, 2, NA, 4, NA))
SumNa(data.frame(a = c(1, NA), b = c(NA, 2)))
SumNa(c(1, NA, "NA", 4), na_list = "NA")
```

`surv_subject_truth` *Collapse Survival Outcomes to One Row per Subject*

Description

Turns right-censored or counting-process (start/stop) survival outcomes into the one-row-per-subject, right-censored truth that yardstick's survival metrics require.

Usage

```
surv_subject_truth(truth, subject_id = NULL)
```

Arguments

`truth`	A [survival::Surv()] object: 'Surv(time, event)' or 'Surv(start, stop, event)'.
`subject_id`	Subject identifiers, one per element of 'truth'. Required for counting-process data; for right-censored data each row is its own subject when 'subject_id' is 'NULL'.

Details

For counting-process data, a subject's time is the stop time of their last interval and their event status is the status of that interval. Rows are checked first: within a subject, intervals must not overlap, and an event may only occur in the last interval (recurrent events are not supported).

Value

A tibble with one row per subject, sorted by 'subject_id': '.subject_id', '.entry' (start of the first interval; 0 for right-censored data), and '.truth', a right-censored 'Surv' object.

See Also

[censoring_km()], [graf_weights()], [add_graf_weights()]

Examples

```
if (requireNamespace("survival", quietly = TRUE)) {
  long <- data.frame(
    subject_id = c(1, 1, 2, 3, 3),
    tstart = c(0, 5, 0, 0, 4),
    tstop = c(5, 9, 6, 4, 10),
    status = c(0, 1, 0, 0, 0)
  )
  surv_subject_truth(
    survival::Surv(long$tstart, long$tstop, long$status),
    subject_id = long$subject_id
  )
}
```

table_two_by_two *Formatted 2x2 Contingency Table with Exact Test*

Description

Builds an institutional 2x2 contingency table with row counts, row percentages, marginal totals, odds ratio estimates, confidence intervals, and exact test p-values. If any cell count is zero, it automatically defaults to the mid-p version of Central Fisher's exact test (`midp = TRUE`).

Usage

```
table_two_by_two(
  data,
  row_var,
  col_var,
  row_label = row_var,
  col_label = col_var,
  midp = NULL,
  conf.level = 0.95
)
```

Arguments

`data`	Data frame containing the categorical variables.
`row_var`	Character name of the row variable.
`col_var`	Character name of the column variable.
`row_label`	Optional display label for the row variable.
`col_label`	Optional display label for the column variable.
`midp`	Logical or `NULL` (default). If `NULL`, defaults to `TRUE` when any cell in the 2x2 table is zero, and `FALSE` otherwise.
`conf.level`	Confidence level for the odds ratio confidence interval (default 0.95).

Value

A list containing:

`table`	Tibble with formatted cells and marginal totals.
`p_value`	Exact test p-value.
`estimate`	Estimated odds ratio.
`conf.int`	Confidence interval for the odds ratio.
`method`	Test method name.
`midp`	Logical indicating whether mid-p adjustment was used.
`has_zero`	Logical indicating whether any table cell had zero count.
`note`	Formatted institutional table note.

temple_brand_path *Path to the Bundled Temple brand.yml*

Description

A copy of the quarto_temple_brand extension's `brand.yml`, for tools that read brand files directly, such as `quarto::theme_brand_ggplot2()` or `bslib::bs_theme(brand = )`.

Usage

```
temple_brand_path()
```

Value

Path to `brand.yml` inside the installed package.

Examples

```
temple_brand_path()
```

temple_colors *Temple University Brand Colors*

Description

Hex codes for the Temple University palette used by the quarto_temple_brand Quarto extension, so R graphics match branded reports. The palette follows Temple's current brand (https://liberalarts.temple.edu/marcom/logos-and-brand): primary cherry, white, and black; secondary Clear Skies and Book Nook; formal accents Academic Gold, Diamond Acres, Founder's Garden, and Night Owl; and casual accents Owl's Eye, Conwell Blue, Upward Momentum, and Cherry Blossom.

Usage

```
temple_colors(...)
```

Arguments

`...`	Optional color names (e.g. `"cherry"`, `"night-owl"`). With none, every color is returned.

Value

A named character vector of hex codes.

See Also

`temple_pal`, `scale_colour_temple`, `theme_temple`

Examples

```
temple_colors()
temple_colors("cherry", "night-owl")
```

temple_pal *Temple Color Palettes*

Description

Returns a palette function that generates n Temple brand colors.

Usage

```
temple_pal(palette = c("main", "diverging", "sequential"), reverse = FALSE)
```

Arguments

palette	One of "main", "diverging", or "sequential".
reverse	Logical; reverse the color order.

Details

"main" Qualitative: cherry, Night Owl, Owl's Eye, Founder's Garden, Upward Momentum, Diamond Acres, black. At most 7 colors.

"diverging" Night Owl through white to cherry, for values centered at zero (correlations, loadings, differences).

"sequential" Book Nook to cherry.

Value

A function taking n and returning n hex codes.

Examples

```
temple_pal()(3)
temple_pal("diverging")(5)
```

theme_cbe *Temple University CBE ggplot2 Themes, Palettes, and Formatters*

Description

Consistent institutional branding and clean publication/presentation styling for Temple University Center for Biostatistics & Epidemiology.

A minimal, publication-ready ggplot2 theme with clean gridlines and bold strip text.

Usage

```
theme_cbe(base_size = 11, base_family = "")
```

Arguments

base_size	Base font size (default: 11)
base_family	Base font family (default: "")

Value

A ggplot2 theme object

---

theme_cbe_deck *CBE ggplot2 Theme for Presentation Decks*

Description

Optimized for slides (PowerPoint/Keynote) with larger typography and bottom legend.

Usage

```
theme_cbe_deck(base_size = 14)
```

Arguments

base_size	Base font size (default: 14)

Value

A ggplot2 theme object

---

theme_temple *Temple ggplot2 Theme*

Description

A minimal theme with Temple cherry titles and facet strips, matching reports rendered with the quarto_temple_brand extension.

Usage

```
theme_temple(base_size = 11, base_family = "")
```

Arguments

base_size	Base font size in points.
base_family	Base font family. The brand's body typeface is `"Faustina"` (headings use `"Roboto"`); the default `""` uses the device font, because naming a font that isn't installed makes devices warn.

Value

A ggplot2 theme.

Examples

```
library(ggplot2)
ggplot(mtcars, aes(wt, mpg)) +
  geom_point(colour = temple_colors("cherry")) +
  facet_wrap(~cyl) +
  labs(title = "Weight vs. mileage") +
  theme_temple()
```

tidy.cbe_loco_mp_coxnet *Tidy a LOCO-MP Cox Model Object*

Description

Extracts feature importance and inference metrics into a clean tibble.

Usage

```
## S3 method for class 'cbe_loco_mp_coxnet'
tidy(x, ...)
```

Arguments

x	A cbe_loco_mp_coxnet object.
...	Not used.

Value

A tibble with term, importance, std_error, statistic, p_value, p_adjusted, conf_low, conf_high. p_value and p_adjusted are one-sided (for importance > 0); conf_low and conf_high are a two-sided interval at level 1 - alpha. See cbe_loco_mp_coxnet.

tidy.coxnet_model *Tidy the Coefficients of a 'coxnet' Model*

Description

Tidy the Coefficients of a 'coxnet' Model

Usage

```
## S3 method for class 'coxnet_model'
tidy(x, penalty = NULL, ...)
```

Arguments

x	A [coxnet()] model.
penalty	The penalty to report coefficients at; defaults to the model's 'penalty'.
...	Not used.

Value

A tibble with 'term', 'estimate' (log hazard ratio per unit of the predictor; 0 for predictors the penalty removed), and 'penalty'.

tidy.joint_model *Tidy Method for Joint Models*

Description

Extracts and aligns nonzero coefficients across the survival (`coxnet`), binary classification (`status`), and duration regression (`time`) models.

Usage

```
## S3 method for class 'joint_model'
tidy(x, ...)
```

Arguments

`x`	A `joint_model` object.
`...`	Additional arguments.

Value

A tibble with comparative coefficients per feature.

tidy_tmerge_cox *Tidy Construction of Counting-Process (Start-Stop) Survival Data*

Description

Merges repeated longitudinal measurement data with event/censoring data (and optional baseline covariates) to build tidy start-stop survival intervals (`tstart`, `tstop`, `event`, `event_label`). This provides a tidy, pipe-friendly alternative to SAS DATA-step macros and `survival::tmerge` for creating counting-process datasets for time-dependent Cox proportional hazards models.

Usage

```
tidy_tmerge_cox(
  measure_df,
  event_df,
  id = "subject_id",
  measure_time = "time",
  event_time = "event_time",
  event_type = "event_type",
  baseline_df = NULL,
  post_event = c("exclude", "include"),
  censor_types = NULL
)
```

Arguments

measure_df	A data frame containing repeated longitudinal measurements.
event_df	A data frame containing subject event or censoring times and event types. It must have one row per subject: repeated ids are an error, because the join would multiply the measurement rows. Subjects that are in event_df but have no measurement are not in the result.
id	Column name identifying subjects across data frames (default: "subject_id").
measure_time	Column name in measure_df indicating measurement observation times (default: "time").
event_time	Column name in event_df indicating event or censoring time (default: "event_time").
event_type	Column name in event_df indicating event label or description (default: "event_type").
baseline_df	Optional data frame containing time-fixed baseline covariates keyed by id. Like event_df, it must have one row per subject.
post_event	Character string: what to do with measurements made at or after a subject's event_time. • "exclude" (default) drops them, so each subject's last interval ends at the recorded event_time. • "include" keeps the follow-up that the later measurements show: when a subject's last measurement is later than event_time, event_time is moved forward to that last measurement time, and the event (or censoring) happens there. The recorded event_time is not kept, so **the survival time of such a subject changes**: with measurements at 0, 3 and 10 and an event_time of 8, the intervals are (0, 3] and (3, 10], with event = 1 on (3, 10] only. The measurement at 10 gives that end point and no interval of its own, so the result never has a zero-length interval (10, 10] or a second event = 1 for the subject. See the examples.
censor_types	Optional character vector of event_type values that mark a censoring time rather than an event, such as "Censored". Those subjects' last interval gets event = 0 (and no event_label). By default (NULL) every subject with an event_time has event = 1 at it, whatever its event_type, so censoring times in event_df must be declared here (or the resulting event recoded) before the data is used to fit a survival model.

Details

Each measurement starts an interval that ends at the subject's next measurement; the last interval ends at event_time, where event is 1 (unless the subject's event_type is in censor_types). A subject therefore needs an event_time, and at least one measurement before it (before the moved event_time, with post_event = "include"), to appear in the result.

Value

A tibble with columns tstart, tstop, event (1 if event occurred at tstop, 0 otherwise), event_label, and all merged measurement and baseline covariates.

Subjects can be missing from the result, or lose their last measurement, and a warning names them: (1) subjects in measure_df without an event_time in event_df (no row, or NA) have no end for their last interval, so that measurement gets no interval, and a subject with a single measurement drops out; (2) subjects with no row in the result at all: with post_event = "exclude" because their event_time is at or before their first measurement, with "include" because all their measurements

are at one time, at or after their `event_time`. A warning also names the subjects with an interval whose `tstop` is not after its `tstart`, which comes from repeated measurement times and which survival models can't use.

See Also

[cbe_cox_multi()]

Examples

```
measures <- data.frame(
  subject_id = c(1, 1, 1),
  time = c(0, 3, 10),
  biomarker = c(1.2, 1.5, 1.8)
)
events <- data.frame(subject_id = 1, event_time = 8, event_type = "Relapse")

# "exclude": the measurement at 10 is after the event and dropped; the last
# interval ends at the event time, (3, 8].
tidy_tmerge_cox(measures, events)

# "include": the event time moves to the last measurement time, so the
# survival time is 10, not 8, and the last interval is (3, 10].
tidy_tmerge_cox(measures, events, post_event = "include")
```

---

tumor_long *Convert Wide Tumor Data to Counting Process (Start/Stop) Format*

Description

Replicates the exact SAS DATA step transformation from SAS/STAT User's Guide (Example 85.7: "Tumor1" dataset). For each subject, intervals (T1, T2] are constructed across observation times where the time-dependent covariate `NPap` changes, with `Status` equal to `Dead` only on the terminal interval and 0 elsewhere.

Usage

```
tumor_long(data = NULL)
```

Arguments

`data`	A data frame structured like `tumor_wide()`. If `NULL`, `tumor_wide()` is used by default.

Value

A tibble with columns:

ID Subject ID.

Time Original death or censoring time.

Dead Original death status.

Dose Dose level.

T1 Start time of the risk interval.

T2 Stop time of the risk interval.

NPap Number of papillomas active during interval (T1, T2].

Status Event status at T2 (1 = event, 0 = censored/continuing).

Source

SAS/STAT User's Guide, Example 85.7: Time-Dependent Repeated Measurements of a Covariate. https://support.sas.com/documentation/cdl/en/statug/68162/HTML/default/statug_phreg_examples07.htm

---

`tumor_wide` *Internal SAS Benchmark Tumor Dataset (Wide Format)*

Description

Transcribes the 45-animal tumor-promoting agent study from the SAS/STAT User's Guide (Example 85.7 / 91.7: "Time-Dependent Repeated Measurements of a Covariate"). In this study, 45 rodents were exposed to a carcinogen and randomized to three dose levels of a tumor-promoting agent (1.0, 2.5, 10.0). The number of papillomas was observed repeatedly across 15 observation times (weeks 27, 34, 37, 41, 43, 45, 46, 47, 49, 50, 51, 53, 65, 67, 71).

Usage

```
tumor_wide()
```

Value

A tibble with 45 rows and 19 columns:

ID Subject ID (1–45).

Time Event or censoring time in weeks.

Dead Censoring status (1 = dead, 0 = censored).

Dose Dose level of the promoting agent (1.0, 2.5, or 10.0).

P1–P15 Number of papillomas counted at observation weeks 27, 34, 37, 41, 43, 45, 46, 47, 49, 50, 51, 53, 65, 67, 71. Missing (NA) beyond death or study termination.

Source

SAS/STAT User's Guide, Example 85.7: Time-Dependent Repeated Measurements of a Covariate. https://support.sas.com/documentation/cdl/en/statug/68162/HTML/default/statug_phreg_examples07.htm

tune_over_alpha *Tune a Penalized Cox Model Over a Grid of 'alpha' Values*

Description

Runs [glmnet_IBS()] on one resampling split for each value in an 'alpha' grid, or for each of a set of candidate formulas, in parallel via [furrr::future_map()]. Set a [future::plan()] first to run in parallel; this function never sets one.

Usage

```
tune_over_alpha(
  object,
  ...,
  num_alpha_values = 10,
  num_fixed = 6,
  alphas = NULL,
  formulas = NULL,
  progress = FALSE
)
```

Arguments

object	An 'rsplit'.
...	Arguments passed to [glmnet_IBS()] ('recipe', 'feature_names', 'time_data', 'id_col', 'cox.ties', ...).
num_alpha_values	Total number of 'alpha' values in the grid.
num_fixed	Number of evenly spaced values from 0 to 1 in the grid.
alphas	Optional numeric vector of 'alpha' values to use instead of the generated grid; in formula mode, one per formula.
formulas	Optional character vector of '+'-separated feature sets, or a list of one-sided formulas, each fit as a separate model (see [glmnet_IBS()]'s 'formula').
progress	Show a progress bar.

Details

Grid mode (the default). The grid is 'num_fixed' evenly spaced values from 0 to 1, plus 'num_alpha_values - num_fixed' values drawn uniformly at random, one per gap between consecutive fixed values (cycling through the gaps if there are more random draws than gaps).

Formula mode ('formulas' given). One fit per formula, each with its own 'alpha': the matching element of 'alphas', or a value drawn uniformly from 0 to 1 when 'alphas' is 'NULL'. Each result gains a 'formula' column.

Random draws use the R session's random number stream, so call [set.seed()] first for reproducible values. Model fits run with 'furrr_options(seed = TRUE)', so they are reproducible too.

To tune 'alpha' and the penalty together in one call, see [cv_coxnet()].

Value

A list named by ‘alpha‘, one element per fit, each the output of [purrr::safely()]: a list with ‘result‘ (the [glmnet_IBS()] tibble, or ‘NULL‘) and ‘error‘ (‘NULL‘, or the condition).

See Also

[glmnet_IBS()], [summarize_tune_results()], [cv_coxnet()]

`tune_race_survival` *Adaptive Racing Tuning for Survival Workflows and Workflow Sets*

Description

Tunes hyperparameters of a survival `workflow` or `workflow_set` using **finetune** racing algorithms (`tune_race_anova` or `tune_race_win_loss`). When given a `workflow_set`, execution is seamlessly mapped across all constituent workflows via `workflow_map`.

Usage

```
tune_race_survival(
  object,
  resamples,
  fn = c("tune_race_anova", "tune_race_win_loss"),
  grid = 20,
  metrics = NULL,
  eval_time = NULL,
  control = NULL,
  seed = 1503,
  ...
)

cbe_tune_race_survival(
  object,
  resamples,
  fn = c("tune_race_anova", "tune_race_win_loss"),
  grid = 20,
  metrics = NULL,
  eval_time = NULL,
  control = NULL,
  seed = 1503,
  ...
)
```

Arguments

`object`	A `workflow` or `workflow_set` specifying the preprocessor and survival model.
`resamples`	An `rset` resampling object (e.g. from `vfold_cv` or `group_vfold_cv`).
`fn`	Character string specifying the racing function: `"tune_race_anova"` (default) or `"tune_race_win_loss"` (which needs **BradleyTerry2**).

grid	Integer number of candidate tuning parameter combinations (default 20), or an explicit parameter grid data frame.
metrics	A metric_set of survival metrics. If NULL (default), defaults to Integrated Brier Score and Concordance.
eval_time	Numeric vector of evaluation time points for dynamic survival metrics. If NULL (default), the deciles of the observed event times in the first resample's data are used, as in cv_coxnet and nested_cv_coxnet; **tune** itself requires at least two evaluation times for the integrated metrics. The outcome is read from the workflow's formula or recipe (for a workflow_set, from its first workflow); pass eval_time for any other preprocessor.
control	A control_race object. Defaults to control_race_survival().
seed	Optional random seed integer for reproducible candidate generation and fold processing.
...	Additional arguments passed to tune_race_anova, tune_race_win_loss, or workflow_map.

Details

Default evaluation metrics are brier_survival_integrated and concordance_survival.

Value

If object is a workflow, a tune_results / race_results object. If object is a workflow set, a workflow_set object containing tuning results.

See Also

control_race_survival, tune_race_anova, workflow_map

Examples

```
## Not run:
if (requireNamespace("finetune", quietly = TRUE) &&
    requireNamespace("survival", quietly = TRUE) &&
    requireNamespace("parsnip", quietly = TRUE) &&
    requireNamespace("workflows", quietly = TRUE) &&
    requireNamespace("rsample", quietly = TRUE)) {
  lung <- stats::na.omit(survival::lung[, c("time", "status", "age", "sex")])
  spec <- parsnip::set_engine(
    parsnip::proportional_hazards(penalty = tune::tune(), mixture = 0.5),
    "coxnet"
  ) |> parsnip::set_mode("censored regression")
  wflow <- workflows::workflow() |>
    workflows::add_model(spec) |>
    workflows::add_formula(survival::Surv(time, status) ~ age + sex)
  folds <- rsample::vfold_cv(lung, v = 4)
  res <- tune_race_survival(wflow, resamples = folds, grid = 10, eval_time = c(180, 365))
  tune::collect_metrics(res)
}

## End(Not run)
```

use_temple_brand *Install the Temple Brand Quarto Extension Into a Project*

Description

Installs quarto_temple_brand with `quarto::quarto_add_extension()`, creating a minimal `_quarto.yml` first if there isn't one (Quarto only applies a brand extension inside a project). Documents in `path` can then use `format: temple-html`, `temple-pdf` (LaTeX title page), `temple-typst`, or `temple-revealjs`, and every format picks up the brand colors, fonts, and logo.

Usage

```
use_temple_brand(
  path = ".",
  extension = "jkylearmstrong-temple/quarto_temple_brand",
  quiet = FALSE,
  check_root = TRUE
)
```

Arguments

`path`	Project directory (created if missing). Defaults to the working directory.
`extension`	Extension source passed to `quarto add`: a GitHub `org/repo`, a URL, or a local path.
`quiet`	Logical; suppress Quarto's output.
`check_root`	Logical; warn when `path` isn't the current project's root as found by here (i.e. the nearest ancestor with a `.Rproj`, `.git`, or similar marker). Set to `FALSE` to install into a report's own subfolder without the warning, e.g. for a one-off report that won't grow siblings.

Details

Install once, at the project root. Quarto resolves a project's `_extensions` from any subfolder of that project, so a single install at the root (where `_quarto.yml` lives) covers every report under it. Calling `use_temple_brand()` again for each report's own subfolder instead creates a separate `_extensions` copy per report, which can drift out of sync as the extension is updated. See create_report's `install_brand` argument for the one-report shortcut, which installs into that report's own folder and is fine when there's only ever going to be one.

Value

Invisibly, a list with `path`, `quarto_yml`, `created_quarto_yml`, and `extension_dir`.

See Also

create_report with `template_name = "temple"`.

Examples

```
## Not run:
# Install once at the project root; every report below it shares the
# extension automatically.
use_temple_brand(here::here())
create_report(here::here("analysis"), template_name = "temple")
create_report(here::here("reports"), template_name = "temple", filename = "q3")

# A single, one-off report: install alongside it and skip the check.
use_temple_brand("analysis", check_root = FALSE)
create_report("analysis", template_name = "temple")

## End(Not run)
```

`validate_column_mapping`

Validate a Plug-and-Play Column Mapping

Description

Verifies that a mapping dictionary has the required schema and logical constraints: columns `INDEX`, `old`, `new`, `X_var`, `Y_var`, `ID_var`, `Time_var`, `duplicate_of`, and `duplicate_action`.

Usage

```
validate_column_mapping(mapping)
```

Arguments

`mapping`	A data frame/tibble containing the column mapping rules.

Value

`TRUE`, invisibly, if valid; otherwise raises an error listing all issues.

`validate_data_manifest`

Validate Data Snapshot Copies Against Upstream Sources

Description

Compares destination files against upstream source files checking: 1. Existence of both source and copy. 2. Cryptographic MD5 hash equality. 3. Object-level equivalence for `.rds` files when MD5 hashes match.

Usage

```
validate_data_manifest(dir = ".", manifest = NULL, project_root = here::here())
```

Arguments

dir	Destination directory containing the snapshot copies.
manifest	Data frame with columns file and source.
project_root	Root directory for resolving source paths (default: here::here()).

Details

Path entries in manifest are validated to ensure they are relative and do not escape dir or project_root.

Value

A tibble with validation status per file.

---

visualize_pipeline *Visualize FilePath dependencies with stage-based coloring and staleness check*

Description

Visualize FilePath dependencies with stage-based coloring and staleness check

Usage

```
visualize_pipeline(
  all_objects,
  extract_graph_code = FALSE,
  stage_colors = NULL
)
```

Arguments

all_objects	The full list of FilePath, FileUses, and FileOutputs objects.
extract_graph_code	Logical. If TRUE, the graph source code will be extracted and returned as a character string.
stage_colors	Optional named list of stage colors. If NULL, uses pipeline_config() options.

visualize_pipeline_interactive

Generate an interactive visNetwork visualization of the computation graph

Description

Generate an interactive visNetwork visualization of the computation graph

Usage

```
visualize_pipeline_interactive(
  all_objects,
  direction = "LR",
  hierarchical = TRUE,
  title = getOption("pipeline.study_name", "Computational Pipeline"),
  subtitle = "Computational Pipeline"
)
```

Arguments

all_objects	A list of FilePath objects or an igraph object.
direction	Layout direction: "LR" (Left-to-Right, default) or "UD" (Top-to-Bottom).
hierarchical	Logical, whether to use a hierarchical DAG layout. Default is TRUE.
title	Character, main title – defaults to getOption("pipeline.study_name", "Computational Pipeline").
subtitle	Character, the document-specific name (e.g. "Computational Pipeline", or a subgraph's own name from export_subgraph()).

Value

A visNetwork htmlwidget.

volcano_plot *Volcano Plot*

Description

Volcano Plot

Usage

```
volcano_plot(
  .data,
  log2_fold_change,
  log_p,
  var,
  alpha = 0.05,
  highlight_significant = TRUE
)
```

Arguments

`.data`	A data frame, e.g. the output of `multiple_t_test`.
`log2_fold_change`	Unquoted column of log2 fold-change.
`log_p`	Unquoted column of `-log10(p.value)`.
`var`	Unquoted column of variable labels.
`alpha`	Significance threshold for the reference line/shading (default `0.05`).
`highlight_significant`	Logical (default `TRUE`); shade the region above the `p < alpha` line.

Value

A ggplot object.

Examples

```
mtcars |>
  dplyr::mutate(am = factor(am)) |>
  multiple_t_test(.class = "am") |>
  volcano_plot(log2_fold_change = log2_fold_change, log_p = log_p, var = var)
```

---

words *Combine Character Vector into Natural-Language Words*

Description

Combine Character Vector into Natural-Language Words

Usage

```
words(x, none = "none")
```

Arguments

`x`	Character vector
`none`	Fallback text when vector is empty (default: "none")

Value

Character string (e.g., "A, B, and C")

write_database_to_excel *Write an R Database or Data Frames to an Excel Workbook*

Description

Writes a named list of data frames (an R database) or a single data frame to a multi-worksheet Excel workbook, with an optional automated data dictionary worksheet.

Usage

```
write_database_to_excel(
  x,
  path,
  include_metadata = TRUE,
  metadata = NULL,
  metadata_sheet = "METADATA",
  ...
)

write_workbook(
  x,
  path,
  include_metadata = TRUE,
  metadata = NULL,
  metadata_sheet = "METADATA",
  ...
)

cbe_write_database(
  x,
  path,
  include_metadata = TRUE,
  metadata = NULL,
  metadata_sheet = "METADATA",
  ...
)
```

Arguments

x	A named list of data frames (an R database) or a single data frame/tibble.
path	File path to save the .xlsx workbook.
include_metadata	Logical; if TRUE (default), computes and appends a data dictionary sheet generated via get_dataset_info.
metadata	Optional pre-computed metadata data frame. If NULL and include_metadata = TRUE, metadata is computed automatically.
metadata_sheet	Character string naming the metadata worksheet (default: "METADATA").
...	Additional arguments passed to write_xlsx.

Value

The file path invisibly.

Examples

```
## Not run:
db <- list(patients = data.frame(id = 1:3), labs = data.frame(id = 1:3, val = 4:6))
write_database_to_excel(db, "clinical_database.xlsx")

## End(Not run)
```

`write_search` *Search a Directory Tree for File-Write Calls*

Description

Convenience wrapper around `find_code` that searches for common data-export function calls (`write_xlsx`, `saveRDS`, `write.csv`, etc.) across a directory tree.

Usage

```
write_search(path, include_comments = FALSE, workers = NULL)
```

Arguments

`path`	Character. Root directory to search.
`include_comments`	Logical. Passed through to `find_code`. Defaults to `FALSE`.
`workers`	Integer number of parallel workers to use when **future/furrr** are installed. Defaults to `NULL`, which caps at available cores minus one. Falls back to sequential search when **future/furrr** are not installed.

Value

A tibble of matches (see `find_code`) with an additional `pattern` column recording which write function matched.

Examples

```
## Not run:
write_search(here::here("analysis"))

## End(Not run)
```

zip_render *Render a Quarto Document and Zip It With Its Dependencies*

Description

Renders `input` in an isolated build directory, copies in any resources it references (explicitly, or heuristically detected from quoted paths / `here::here()` calls), and zips the outputs together with the source and sidecar files.

Usage

```
zip_render(
  input,
  formats = c("html", "pdf", "docx"),
  resources = NULL,
  detect = c("heuristic", "none"),
  build_dir = NULL,
  zip_name = NULL,
  copy_back_dir = NULL,
  include_sources = TRUE,
  overwrite = TRUE,
  verbose = TRUE
)
```

Arguments

`input`	Path to the input `.qmd` file.
`formats`	Character vector of output formats (e.g. `c("html","pdf","docx")` or `"all"`).
`resources`	Optional character vector of extra files to include, absolute or project-relative.
`detect`	`"heuristic"` (default; scans the `.qmd` for likely file paths) or `"none"`.
`build_dir`	Staging directory; defaults to a fresh temp directory. A relative path is relative to the working directory the function is called from.
`zip_name`	Name of the resulting zip; defaults to `<input-stem>.zip`.
`copy_back_dir`	Where to copy the finished zip; defaults to `dirname(input)`. A relative path is relative to the working directory the function is called from.
`include_sources`	Logical (default `TRUE`); include the `.qmd` and sidecar bib/tex/css files.
`overwrite`	Logical (default `TRUE`); overwrite an existing zip at the destination.
`verbose`	Logical (default `TRUE`); print progress messages.

Details

The rendered outputs are the files in the build directory named `<input-stem>.<extension>` for the extensions of `formats` (the case of the name is ignored). If there are none, for instance because the format writes its output to another folder, a warning says so and the zip holds the sources only.

Heuristic detection inspects quoted file paths and here::here() calls in the document. For YAML front-matter resources, pass them explicitly via the ‘resources‘ argument or include them in the document’s YAML; this function will attempt to parse YAML when present to pick up top-level resource lists.

Value

Invisibly, a list with the build directory, detected/copied resources, render outputs, and final zip path.

Examples

```
## Not run:
zip_render("report.qmd", formats = c("html", "pdf"))

## End(Not run)
```

`zip_reports` *Package Multiple Already-Rendered Reports Into an Indexed Zip*

Description

Given a data frame describing an ordered set of already-rendered Quarto reports grouped into stage folders, copies their existing PDF/DOCX/HTML outputs into a staged build directory, builds an Excel index with hyperlinks to each report, optionally bundles arbitrary data-deliverable files alongside them, and zips the result.

Usage

```
zip_reports(
  reports,
  output_formats = c("pdf", "docx"),
  data_deliverables = character(0),
  zip_name = NULL,
  output_dir = getwd(),
  docx_from_pdf = NULL
)
```

Arguments

`reports`	A data frame/tibble with one row per report and columns `name` (display name), `path` (path to the `.qmd` source — the corresponding `.pdf`/`.docx`/`.html` output paths are derived by swapping its extension), and optionally `stage` (subfolder to group the report under; `NA`/empty becomes `"99_Other"`) and `description`. The stage is used as a folder name inside the zip, so path separators and the characters a file name cannot hold on Windows become `-` and leading or trailing dots, dashes and spaces are dropped (`"../x"` gives `"x"`); ordinary names such as `"01 Results"` are kept as they are. The index shows the stage as it was given.
`output_formats`	Character vector of formats to look for and include: any of `"pdf"`, `"docx"`, `"html"`, or `"all"` for all three. Defaults to `c("pdf", "docx")`.
`data_deliverables`	Character vector of additional file paths to copy into a top-level `data/` folder in the zip. Defaults to `character(0)`. A path that does not exist is skipped. Files that share a file name (`a/results.csv`, `b/results.csv`) are all kept, the later ones with a numeric suffix (`results_2.csv`) and a warning. A file that cannot be copied (open in another program, a folder instead of a file) is left out with a warning.

zip_name	Name of the output zip file. Defaults to all_reports_<date>.zip.
output_dir	Directory the finished zip is written to. Defaults to getwd().
docx_from_pdf	Optional function function(src_pdf, dest_docx) used to create dest_docx from an existing src_pdf when a report requests "docx" output but no (or a stale) .docx file exists next to its .qmd source. A DOCX is stale when it is older than its PDF. If NULL (default), a missing DOCX is skipped silently and a stale one is skipped with a warning; it is never shipped. The function counts as having failed when it returns FALSE (as convert_pdf_to_docx does) or when no DOCX at least as new as the PDF is there afterwards; a warning says so and no DOCX is shipped for that report. Any other return value is taken as success. In the index, the DOCX cell of a report whose PDF exists but that has no DOCX to ship reads "not converted".

Details

Unlike zip_render (which renders and zips a single document), zip_reports does not render anything itself — it packages outputs that have already been rendered, e.g. by render (or render_me). Report order and staging is entirely the caller's decision (for example, the topological sort of a project's own dependency graph) — pass reports pre-ordered. If multiple reports share the same source stem (for example analysis1/analysis.qmd and analysis2/analysis.qmd), staged output names are disambiguated to prevent overwrites within a stage/format folder.

Value

The path to the created zip file.

Examples

```
## Not run:
reports <- data.frame(
  name = c("Introduction", "Results"),
  path = c("analysis/intro.qmd", "analysis/results.qmd"),
  stage = c("00_intro", "01_results"),
  description = c("Project overview", "Primary analysis results")
)
zip_reports(reports, output_formats = c("pdf", "docx"))

## End(Not run)
```

---

z_norm *Z-Score Standard Normalization*

Description

Standardizes numeric features to have mean = 0 and standard deviation = 1. For a matrix or data frame each (numeric) column is standardized on its own. A constant column (or a lone observation) has no spread to divide by and comes back as 0, with NAs kept as NA. With na.rm = FALSE, a column containing an NA has no defined mean and comes back all NA.

Usage

```
z_norm(x, na.rm = TRUE)
```

Arguments

x	A numeric vector, matrix, or data frame.
na.rm	Logical; whether to ignore NA values (default TRUE).

Value

Z-score standardized numeric object.

Examples

```
z_norm(c(10, 20, 30, 40, 50))
```

Index

%!in% (infix_helpers), 128
%flike% (infix_helpers), 128
%ilike% (infix_helpers), 128
%like% (infix_helpers), 128
%notin% (infix_helpers), 128
%plike% (infix_helpers), 128

add_graf_weights, 8
add_missing, 9
add_p, *38*, *57*
anonymize_pi, 10, *168*
apply_database_metadata, 12
as.package, *18*
as_database, 13, *23*, *93–95*, *126, 127*
as_igraph, *13*, *95*, *122*, *134*
as_igraph (as_igraph.cbe_database_relationships), 14
as_igraph.cbe_database_relationships, 14
as_pipeline_graph, 14, *15*
as_tbl_graph.cbe_database_relationships (cbe_database_relationships), 32
as_tbl_graph.list, 15
audit_report_deliverables, 15
autoplot.cbe_compare_df, 16
autoplot.cbe_database_relationships, 16
autoplot.cbe_key_integrity, 17
autoplot.cbe_loco_mp_coxnet, 17
autoplot.cv_coxnet (cv_coxnet-methods), 90
autoplot.cv_joint_model, 18

balloon_plot (cbe_contingency_plot), 24
bar_plot (cbe_contingency_plot), 24
brier_survival_integrated, *224*
build_manual_versioned, 18

calculate_fences, 19
cbe_balloon_plot, 19
cbe_bar_plot, 20
cbe_check_key_integrity, *17*
cbe_check_key_integrity (cbe_database_relationships), 32
cbe_compare_df, *13*, *16*, 21, *32, 33*, *94*, *127*
cbe_contingency_plot, *19, 20*, 24, *44*, *50*, *54*
cbe_control_race (control_race_survival), 70
cbe_cox_check, 28
cbe_cox_multi, 29
cbe_cox_single, 30
cbe_cox_table, 31
cbe_database, *23*, *93*, *95*
cbe_database (as_database), 13
cbe_database_label (cbe_labels), 46
cbe_database_label<- (cbe_labels), 46
cbe_database_name (cbe_labels), 46
cbe_database_name<- (cbe_labels), 46
cbe_database_relationships, *13, 14*, *16*, 32
cbe_database_venn, 33
cbe_dataset_label (cbe_labels), 46
cbe_dataset_label<- (cbe_labels), 46
cbe_docx_review_extract, 34, *192, 193*
cbe_exact2x2, 37, *39*, *43*, *54*, *170*
cbe_exact2x2_ci, 39
cbe_explain, 40
cbe_explain_stack (cbe_explain), 40
cbe_explain_status (cbe_explain), 40
cbe_explain_survival, 41
cbe_explain_time (cbe_explain), 40
cbe_factor_reference, 42
cbe_find_shared_keys (cbe_database_relationships), 32
cbe_four_quadrant_report, 43
cbe_get_dataset_labels (cbe_labels), 46
cbe_get_id_cols (cbe_variable_roles), 59
cbe_get_outcomes (cbe_variable_roles), 59
cbe_get_predictors (cbe_variable_roles), 59
cbe_get_roles (cbe_variable_roles), 59
cbe_get_time_cols (cbe_variable_roles),

59
cbe_heatmap_plot, 44
cbe_km_single, 45
cbe_labels, 46
cbe_loco_mp_coxnet, 47, *217*
cbe_mosaic_plot, 50
cbe_pairwise_combos, 51
cbe_palette, 52
cbe_pformat *(*pformat*)*, 166
cbe_predict_parts_shap, 52
cbe_read_database
*(*read_database_from_excel*)*, 185
cbe_sas_macro_dir, 53
cbe_sas_macro_path, 54
cbe_set_dataset_labels *(*cbe_labels*)*, 46
cbe_set_roles *(*cbe_variable_roles*)*, 59
cbe_square_plot, 54
cbe_survex_loss_brier
*(*cbe_survex_loss_ibs*)*, 56
cbe_survex_loss_ibs, 56
cbe_test_categorical, 57
cbe_theme_survival, 59
cbe_tune_race_survival
*(*tune_race_survival*)*, 223
cbe_variable_roles, 59, *61*
cbe_write_database
*(*write_database_to_excel*)*, 230
censoring_km, 61
check_docx_toolchain, 62
check_dots_empty, *179*
check_key_integrity
*(*cbe_database_relationships*)*,
32
chisq.test, *27*, *44*, *55*, *58*
clean_names, 63, *187*, *197*
clean_publish, 63, *128*
collapse_by_stage, 68
collect_metrics.cv_coxnet
*(*cv_coxnet-methods*)*, 90
collect_metrics.nested_cv_coxnet
*(*nested_cv_coxnet*)*, 151
combn, *79*
compare_df *(*cbe_compare_df*)*, 21
compute_graph, *13*, 69, *94*
concordance_survival, *224*
contingency_plot
*(*cbe_contingency_plot*)*, 24
control_race, *70*, *71*, *184*, *224*
control_race_survival, 70, *224*
controls_datasets, 69
convert_pdf_to_docx, 72, *234*
convert_pdfs_to_docx, 71
copy_data_manifest, 73
cor, *76*, *109*
cor.test, *78*, *79*
corr_test_all, 78
correlation_diff, 74, *75*
correlation_diff_heatmap, 75
correlation_plot, *74*, *75*, 76, *77*, *78*, *169*
correlation_plot_split, 77, *77*
corrplot, *76*, *77*, *170*
cox_check *(*cbe_cox_check*)*, 28
cox_multi *(*cbe_cox_multi*)*, 29
cox_single *(*cbe_cox_single*)*, 30
cox_table *(*cbe_cox_table*)*, 31
coxnet, 79, *100*, *132*, *133*, *184*
coxnet_train, 81
create_qmd_renderer, 83, *104*, *106*, *167*,
191
create_report, 83, *225*
create_toc_from_sas_pdf, 85
cutree, *77*, *78*
cv.glmnet, *133*
cv_coxnet, *49*, 86, *91*, *92*, *132*, *133*, *154*, *224*
cv_coxnet-methods, 90
cv_joint_model, 91, *133*, *154*, *155*, *179*

database_label *(*cbe_labels*)*, 46
database_label<- *(*cbe_labels*)*, 46
database_metadata_io, 92
database_name *(*cbe_labels*)*, 46
database_name<- *(*cbe_labels*)*, 46
database_relationships
*(*cbe_database_relationships*)*,
32
database_setops, *13*, 93, *95*, *126*
database_to_igraph, *93*, 95, *126*
database_venn *(*cbe_database_venn*)*, 33
dataset_label *(*cbe_labels*)*, 46
dataset_label<- *(*cbe_labels*)*, 46
delete_nul_files, 96
demo_cbe_mapping, 96
detect_outliers, 97
discrete_scale, *198*
distribution_plot, 97
distribution_test, 98
docx_review_extract
*(*cbe_docx_review_extract*)*, 34

exact2x2, *38*, *58*
exact2x2 *(*cbe_exact2x2*)*, 37
exact2x2_ci *(*cbe_exact2x2_ci*)*, 39
explain_joint *(*cbe_explain*)*, 40
explain_stack *(*cbe_explain*)*, 40
explain_status *(*cbe_explain*)*, 40

explain_survival
(cbe_explain_survival), 41
explain_time (cbe_explain), 40
export_interactive_pipeline, *69*, 98
export_subgraph, *69*, 99
extract_all_xlsx_tokens, 100
extract_surv_components, 100
extract_win_posix_paths, 101
extract_yaml_formats, 102

factor_reference
(cbe_factor_reference), 42
features_percent_miss, *9*, 102, *169*
file_meta_fs, 107
FileOutputs, 103, *103*, *104*, *106*, *107*
FileOutputs-class, 103
FilePath, *103*, 104, *104–107*
FilePath-class, 105
FileUses, *104*, 106, *106*, *107*
FileUses-class, 107
find_code, 108, *189*, *231*
find_correlation, 109
find_python, 110, *111*
find_sas, *110*, 111
find_section_file, 112
find_shared_keys
(cbe_database_relationships), 32
find_soffice, 112
findCorrelation, *109*
fisher.test, *27*, *44*, *55*, *58*
fit_best, *70*
fit_n_rank, 113
flag_outliers, *97*, 114
flike (infix_helpers), 128
fmt_hr, 114
fmt_num, 115
fmt_p, 115
fmt_pct, 116
fmt_sig, 116
forge, *41*, *101*, *133*, *179*
formula, *58*
four_quadrant_report
(cbe_contingency_plot), 24

generate_last_names, *10*, *11*, 117, *168*
generate_pi_names, 117, *168*
generate_pseudonym_token, 118, *168*
get_dataset_info, *93*, 120, *230*
get_dataset_labels (cbe_labels), 46
get_id_cols (cbe_variable_roles), 59
get_model_parameters, 121
get_outcomes (cbe_variable_roles), 59
get_predictors (cbe_variable_roles), 59
get_render_plan, *14*, 122, *191*
get_roles (cbe_variable_roles), 59
get_time_cols (cbe_variable_roles), 59
ggplot, *16*, *33*
ggVennDiagram, *33*
glmnet, *49*, *133*
glmnet_IBS, *91*, 123, *133*
graf_weights, 125
graph_intersect, *93*
graph_intersect (graph_setops), 126
graph_setops, 126
graph_subtract, *93*
graph_subtract (graph_setops), 126
graph_union, *93*
graph_union (graph_setops), 126
grep, *108*
grepl, *129*
group_bootstraps, *92*
group_vfold_cv, *91*, *92*, *223*
guard_public_remote, *67*, 127

heatmap_plot (cbe_contingency_plot), 24
here, *225*

ilike (infix_helpers), 128
infix_helpers, 128
install.packages, *129*
install.packages.no_lock, 129
is.int, 130
is_normal, *98*, 130, *131*
is_poisson, *98*, 131

join_pipelines, *94*, *127*, 134
joint_model, *91*, *120*, *121*, 132, *154*, *155*, *179*, *207*

keep_only, 135
km_single (cbe_km_single), 45
km_summary_to_prism, 135

label_pvalue, *166*
like (infix_helpers), 128
list_stage_data_artifacts, 137
loco_mp_coxnet (cbe_loco_mp_coxnet), 47

make_excel_names, 137
manhattan_plot, 138
metric_set, *224*
min_max_norm, *97*, 138, *185*
missforest_impute_by_mtry, 139
missforest_oob_by_mtry, 140
missforest_sweep_mtry, 141
missmap, 143

missranger_max_mtry, 144
missranger_oob_by_mtry, 145
missranger_sweep_mtry, 147
mosaic_plot (cbe_contingency_plot), 24
multiple_t_test, *138*, 150, *229*
my_summary_table, 151

nested_cv, *154*
nested_cv_coxnet, 151, *224*
nested_cv_joint_model, *133*, 154
normalize_safely, 155
notin (infix_helpers), 128

one_vs_rest_t_test, 156

package_deliverables, 157
pairwise_combos (cbe_pairwise_combos), 51
parse_here_call_vec, 158
pca_biplot, 158, *163*
pca_eqns, 159, *195*
pca_feature_loading_heatmap, 160, *161*, *163*
pca_loading_diff, *74*, 160, *161*
pca_loading_diff_heatmap, *75*, 161
pca_loadings (rotation_matrix), 195
pca_percent_var_explained, 162, *163*
pca_plot, 162
pca_scree_plot, *162*, 163, *163*
pca_variables_plot, *163*, 164
pdf_to_rtf, 165
pformat, *27*, *44*, *55*, 166
pi_anonymizer, *11*, *118–120*, 168
pipeline_config, 167, *191*, *192*
pipeline_summary, 167, *211*
plan, *58*
plike (infix_helpers), 128
plot.features_percent_miss, 169
plot_categorical_associations, 169
plot_cox_forest, 171
plot_cox_forest_multi, 172
plot_cox_marginal, 173
plot_cox_survival, 174
plot_dynamic_trajectory, 175
plot_features_percent_miss (plot.features_percent_miss), 169
plot_group_comparison, 175
plot_missingness, 176
plot_pca_bi, *163*, 177
plot_survival_km, 177
prcomp, *159–164*, *177*, *183*, *195*
predict.censoring_km (censoring_km), 61
predict.coxnet_model, 178
predict.cv_coxnet (cv_coxnet-methods), 90
predict.joint_model, *41*, *133*, 179
predict_coxnet_linear_pred (coxnet_train), 81
predict_coxnet_survival (coxnet_train), 81
predict_coxnet_time (coxnet_train), 81
predict_parts_shap (cbe_predict_parts_shap), 52
print.cbe_cox, 180
print.cbe_cox_check, 180
print.cbe_cox_multi, 181
print.cbe_km, 181
print.joint_model, 181
print_pipeline, *69*, 182
proc_contents (get_dataset_info), 120
proc_pca, *163*, *164*, 183
profvis_summary, 183

R_names, 197
racing_workflows, 184
range_norm, *97*, 185
read_data_manifest, 186
read_database_from_excel, 185
read_database_metadata (database_metadata_io), 92
read_excel, *186*, *187*, *190*
read_excel_multiple_headers, 187, *189*
read_mapped_section_data, 188, *209*
read_raw_table, 189
read_search, 189, *199*, *200*
read_workbook, *13*, *93*, *189*, 190
render, *167*, 190, *234*
render_me, *234*
render_me (render), 190
required_pkgs.step_lencode_coxnet (step_lencode_coxnet), 205
required_pkgs.step_lencode_joint_model (step_lencode_joint_model), 207
review_config, *35*, 192
rossi_data, 194
rotation_matrix, 195
run_sas_script, 196

sas_available (find_sas), 111
sas_macro_dir (cbe_sas_macro_dir), 53
sas_macro_path (cbe_sas_macro_path), 54
scale_color_cbe, 197
scale_color_temple (scale_colour_temple), 198
scale_colour_gradient2, *198*

scale_colour_gradientn, *198*
scale_colour_temple, 198, *214*
scale_fill_cbe, 199
scale_fill_temple *(scale_colour_temple)*, 198
scan_data_io, 199
sd.error, 200
set_dataset_labels *(cbe_labels)*, 46
set_roles *(cbe_variable_roles)*, 59
show,FileOutputs-method *(FileOutputs-class)*, 103
show,FilePath-method *(FilePath-class)*, 105
show,FileUses-method *(FileUses-class)*, 107
significance_stars, 201
simulate_section_data, 201
single_t_test, *150*, *156*, 202
square_plot *(cbe_contingency_plot)*, 24
step_famd, 203
step_lencode_coxnet, 205, *207, 208*
step_lencode_joint_model, *206*, 207
stop_if_invalid_manifest, 209
summarize_pipeline *(pipeline_summary)*, 167
summarize_section_by_time, 209
summarize_tune_results, 210
summary.tbl_graph, 211
SumNa, *102*, 211
Surv, *132*
surv_subject_truth, 212
survex_loss_brier *(cbe_survex_loss_ibs)*, 56
survex_loss_ibs *(cbe_survex_loss_ibs)*, 56

t.test, *150*, *203*
table_two_by_two, 213
tbl_summary, *210*
temple_brand_path, 214
temple_colors, 214
temple_pal, *214*, 215
test_categorical *(cbe_test_categorical)*, 57
tests, *38*, *58*
theme_cbe, 215
theme_cbe_deck, 216
theme_temple, *214*, 216
tidy.cbe_loco_mp_coxnet, 217
tidy.coxnet_model, 217
tidy.cv_coxnet *(cv_coxnet-methods)*, 90
tidy.joint_model, 218
tidy.step_famd *(step_famd)*, 203
tidy_tmerge_cox, 218
tumor_long, 220
tumor_wide, 221
tunable.step_lencode_coxnet *(step_lencode_coxnet)*, 205
tune_over_alpha, 222
tune_race_anova, *155*, *184*, *223, 224*
tune_race_survival, *71*, 223
tune_race_win_loss, *155*, *184*, *223, 224*

use_temple_brand, *84*, 225

validate_column_mapping, *59*, *202*, 226
validate_data_manifest, *74*, *209*, 226
var_label, *197*
variable_roles *(cbe_variable_roles)*, 59
vfold_cv, *91, 92*, *223*
visualize_pipeline, *14*, *69*, 227
visualize_pipeline_interactive, *69*, 228
volcano_plot, 228

words, 229
workflow, *223*
workflow_map, *223, 224*
workflow_set, *223*
write_database_metadata *(database_metadata_io)*, 92
write_database_to_excel, 230
write_search, *199, 200*, 231
write_workbook, *13*
write_workbook *(write_database_to_excel)*, 230
write_xlsx, *230*

z_norm, *97*, 234
zip_render, 232, *234*
zip_reports, 233